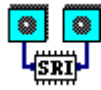


SRI Small Vision System



User's Manual
Software version 2.2f
January 2002

©Kurt Konolige and David Beymer
SRI International
konolige@ai.sri.com
<http://www.ai.sri.com/~konolige>

1 Introduction	5
1.1 The SRI Stereo Engine and the Small Vision System	6
1.2 The Small Vision System	7
1.3 Hardware and Software Requirements	8
1.3.1 Analog Framegrabbers	8
1.3.2 Digital Framegrabbers	8
1.4 The SVS Distribution	9
2 Getting started with <i>smallv</i>	10
2.1 Inputting Live Stereo Video	12
2.1.1 Stereo heads	12
2.1.2 Analog Framegrabbers	13
2.1.3 IEEE 1394 (FireWire) Framegrabber	14
2.1.4 Selecting Devices	14
2.1.5 Frame Size	15
2.1.6 Image Sampling	16
2.1.7 Image Source	16
2.1.8 Streaming Mode	16
2.1.9 Adjusting Video Parameters	17
2.1.10 Subwindowing	18
2.1.11 Vergence	19
2.1.12 Color	20
2.2 Storing, Saving, and Loading Stereo Data	21
2.2.1 Stereo Stream Storage	21
2.2.2 Loading and Storing Files	21
2.3 Display	23
2.4 Stereo Processing and Parameters	24
2.4.1 Stereo Function	24
2.4.2 3D Transformation	24
2.4.3 Calibration	26
2.4.4 Disparity Search Range	26
2.4.5 Adjusting the Horopter	27
2.4.6 Pixel Information	27
2.4.7 Correlation Window Size	27
2.4.8 Multiscale Disparity	28
2.5 Filtering	29
2.5.1 Confidence Filter	29
2.5.2 Left/Right Filter	29
2.6 Saving and Restoring Parameters	30
3 Stereo Geometry	32
3.1 Disparity	33
3.2 Horopter	35
3.3 Range Resolution	38
3.4 Area Correlation Window	39

3.5 Multiscale Disparity	41
3.6 Filtering	42
3.7 Performance	44
4 Calibration	45
4.1 Calibration Procedure	46
4.1.1 Calibration procedure steps	46
4.1.2 Calibration Target	48
4.1.3 Imager Characteristics	48
5 API Reference – C++ Language	49
5.1 Threading and Multiple Stereo Devices	50
5.1.1 Threading Issues	50
5.1.2 Multiple Devices	50
5.2 C++ Classes	51
5.3 Parameter Classes	54
5.3.1 Class svImageParams	54
5.3.2 Class svRectParams	54
5.3.3 Class svDispParams	54
5.4 Stereo Image Class	55
5.4.1 Constructor and Destructor	55
5.4.2 Stereo Images and Parameters	55
5.4.3 Rectification Information	56
5.4.4 Disparity Image	56
5.4.5 3D Point Array	56
5.4.6 File I/O	57
5.4.7 Copying Functions	57
5.5 Acquisition Classes	58
5.5.1 Constructor and Destructor	58
5.5.2 Rectification	58
5.5.3 Controlling the Image Stream	58
5.5.4 Error String	59
5.6 Video Acquisition	60
5.6.1 Video Object	60
5.6.2 Device Enumeration	60
5.6.3 Opening and Closing	60
5.6.4 Image Framing Parameters	61
5.6.5 Image Quality Parameters	62
5.6.6 Controlling the Video Stream	62
5.7 File and Memory Acquisition	63
5.7.1 File Image Object	63
5.7.2 Setting Images from Files	63
5.7.3 Stored Image Object	63
5.7.4 Setting Images from Memory	63
5.8 Stereo Processing Classes	65
5.8.1 Stereo and 3D Processing	65
5.8.2 Multiscale Stereo Processing	65
5.9 Window Drawing Classes	66

5.9.1	Class svWindow	66
5.9.2	Class svDebugWin	67

1 Introduction

The SRI Stereo Engine is an efficient realization of an area correlation algorithm for computing range from stereo images. Figure 1 shows the results of running the algorithm on a typical scene. The image on the top left is the left image of an original stereo pair, while the one on the top right is a disparity image computed from the stereo pair. In the disparity images, brighter pixels show where the projection of an object diverges between the images (has a high disparity). These are areas that are closer to the cameras. Dark areas have lower disparity, and are further away. Finally, the bottom right shows a view of the 3D reconstruction made from the disparity image.

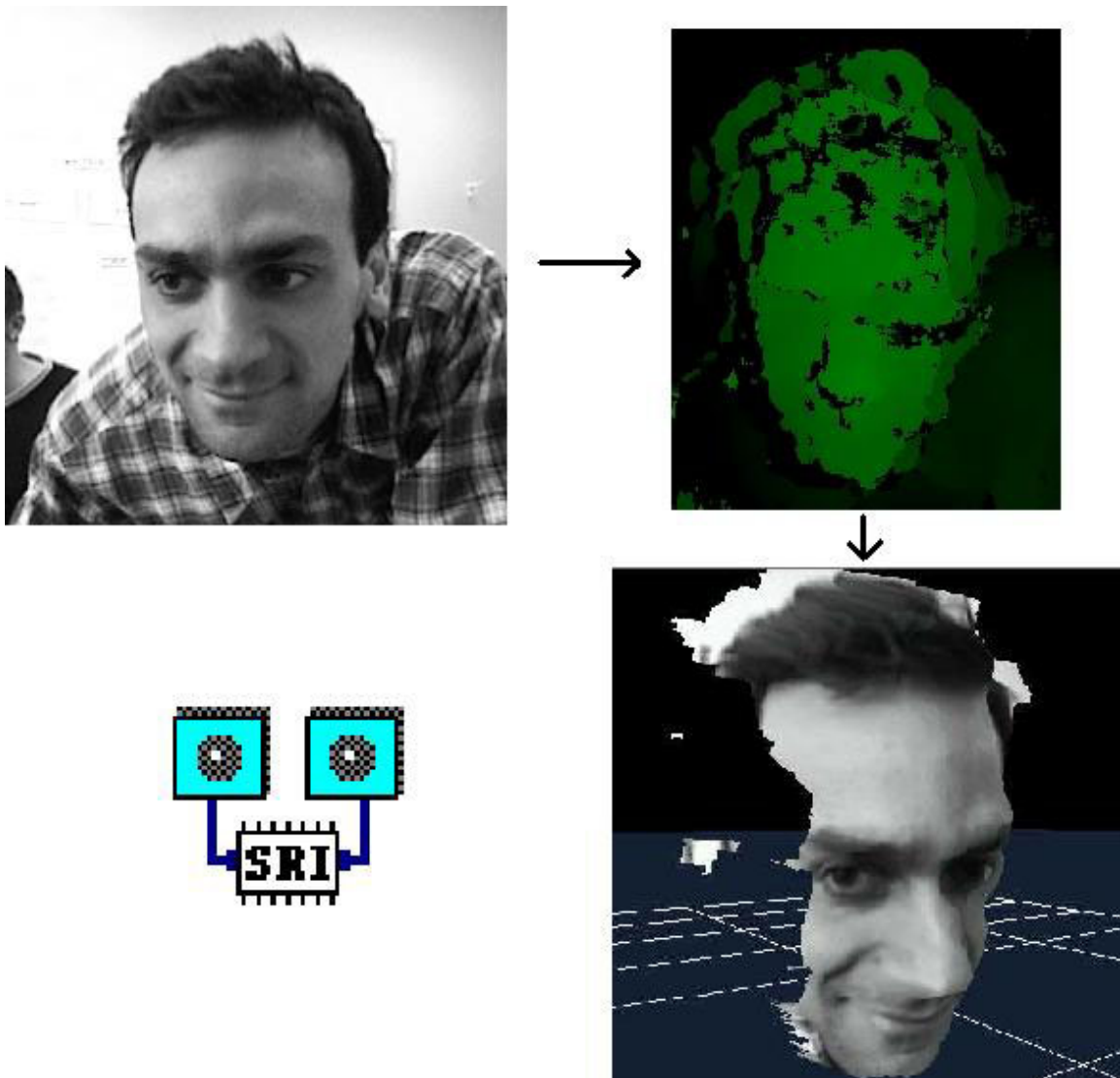


Figure 1-1. An input image and the resultant stereo disparity image. Brighter areas are closer to the camera.

1.1 The SRI Stereo Engine and the Small Vision System

The Stereo Engine exists in several implementations, including embedded, low-power systems and general purpose microcomputers. The embedded systems, or Small Vision Modules (SVMs), contain DSPs or other standalone processors, and produce digital range information. They are meant for end applications where size, cost, and power limitations are critical. SRI will develop embedded SVM systems in partnership with companies who are interested in a particular application.

The Small Vision System (SVS) is an implementation of the Stereo Engine on general-purpose microcomputers, especially PCs running Linux or Windows 95/98/ME/2000/NT. It consists of a set of library functions implementing the stereo algorithms. Users may call these functions to compute stereo results on any images that are available in the PC's memory. Typically, standard cameras and video capture devices are used to input stereo images. The Small Vision System is a development environment for users who wish to explore the possibility of using stereo in an application.

This manual is useful as a source of general information about the Stereo Engine for any implementation, but is also specifically aimed at the development environment of the SVS. It explains the core characteristics of the Stereo Engine, serves as a reference for the stereo function API, and discusses sample applications that use the API. There are also several tutorials that illustrate writing programs to the SVS API, in the documentation folder. More technical information about stereo processing can be found at www.ai.sri.com/~konolige/svs, including several papers about the stereo algorithms and applications.

With Version 2.2x of SVS, we introduce a re-written C++ interface to the SVS libraries. The new C++ classes are much simpler to use than the previous C functions, and in particular relieve the user of having to perform buffer management for images and disparity results. The C function interface is still maintained for users who have an investment in application programs, under the 2.1x versions of SVS, but we will only upgrade this version with bug fixes. Users are encouraged to migrate to the 2.2x API.

1.2 The Small Vision System

The Small Vision System (SVS) is meant to be an accessible development environment for experimenting with applications for stereo processing. It consists of a library of functions for performing stereo correlation. Figure 1-2 shows the relationship between the SVS library and PC hardware.

Images come in via a pair of aligned video cameras, called a stereo head. A video capture board or boards in the PC digitizes the video streams into main memory. The SVS functions are then invoked, and given a stereo pair as an argument. These functions compute a disparity image, which the user can display or process further.

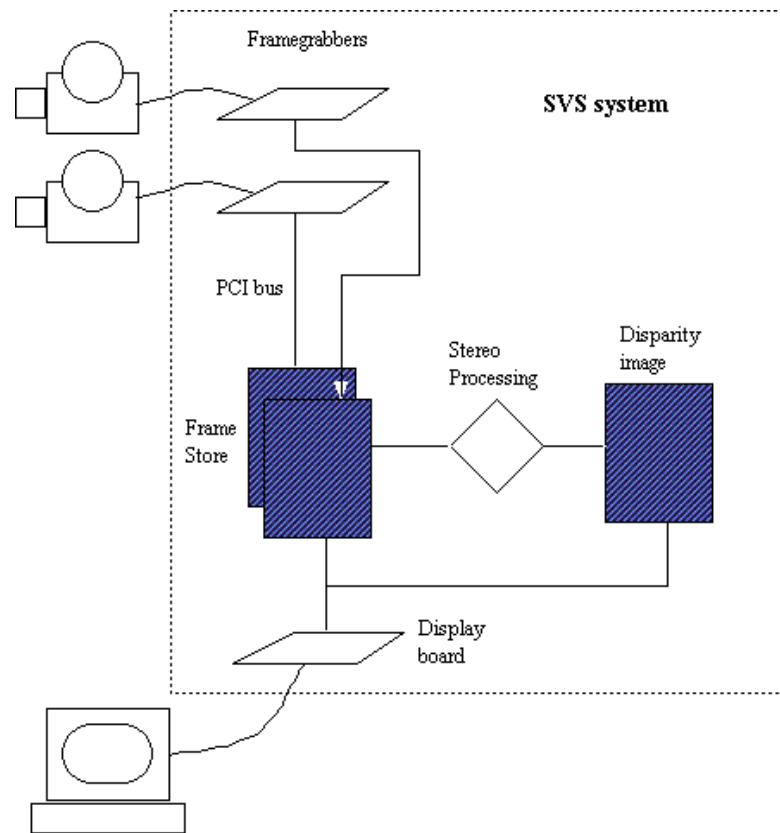


Figure 1-2. The development environment of the Small Vision System.

The SVS environment of Figure 1-2 shows a typical setup for stereo processing of video images. The user may supply his or her own cameras: the SVS has special processing for dealing with camera distortion and calibrating the stereo image (Section 2.4.2). Special stereo heads are also available from Videre Design (www.videredesign.com). The STH-V3 is an analog head with the ability to send a stereo pair on just a single video signal, so only one video capture device is required (more information on video capture is in Section 2.1). The MEGA-D (STH-MD1) is an all-digital device with megapixel imagers that uses the 1394 bus (FireWire) for direct digital input. Finally, other sources of images may also be used, as long as the images can be placed in PC memory. Some examples are images stored on disk, or images obtained from other devices such as scanning electron microscopes.

1.3 Hardware and Software Requirements

The SVS libraries exist for most Unix systems, as well as MS Windows 95/98/ME/2000/NT; that is, on the most common computer platforms available. We have spent considerable effort in optimizing SVS for PCs using the MMX instruction set (Pentium MMX and Pentium II, III, IV), and it will perform best on these platforms, using either Linux or MS Windows. Performance information is in Section 3.7.

1.3.1 Analog Framegrabbers

Because of the ubiquity of PCs, we have added support for several PCI bus video capture devices on PCs. The following are recommended PC hardware configurations for the SVS with analog cameras.

Operating System	Video Capture Card
Linux	Imagenation PCX200 Matrox Meteor and Meteor RGB Any Bt848-based card (e.g., Intel Smart Video Recorder III)
Windows 95/98/ME	Imagenation PXC200 Matrox Meteor, Meteor RGB, Meteor II
Windows NT, 2000	Imagenation PXC200 Matrox Meteor, Meteor RGB, Meteor II

Table 1-1. Analog Framegrabbers and OS requirements for the Small Vision System.

Unfortunately, as of this time there are no good analog framegrabbers for laptops. The MRT Video Port Pro is one of the best cards, but it is still slow for input to memory, and does not take advantage of the 32-bit CardBus specification. However, the MEGA-D digital device (see below) does have laptop input capability.

1.3.2 Digital Framegrabbers

The MEGA-D (STH-MD1) and Dual DCAM (STH-DCAM) are all-digital devices that use the IEEE 1394 (FireWire) bus. Some desktops and laptops have 1394 ports integrated directly into their motherboards. Otherwise, a standard 1394 PCI board or PCMCIA card can be used. The card must be OHCI (Open Host Controller Interface) compliant, which almost all boards are.

1.4 The SVS Distribution

The SVS distribution can reside in any directory; normally, it is placed in `c:\svs` (MS Windows systems) or `/usr/local/svs` or a user's home directory (Unix systems). Here is the directory structure of the SVS distribution.

svs	
readme	installation guide
update	release notes
docs	documentation
smallv.pdf	PDF version of the User Manual
calibrate.pdf	PDF version of the Calibration Addendum
bin	executable and library files
smallv(.exe)	full-featured GUI client demo
stframe(.exe)	simple stereo client example program
stcap(.exe)	frame capture program, no stereo processing
XXXgrab.dll, .lib	framegrabber interface fns (Windows OS)
svs.dll, lib	SVS library (Windows OS)
svscap.dll, lib	SVS library for capture only (Windows OS)
libsvs.so	SVS library (Linux)
libsvscap.so	SVS library for capture only (Linux)
fltkdll.dll	Display library (Windows OS)
libfltk.so	Display library (Linux)
data	stereo images
check.pdf	single-page printable calibration object
check2.pdf	4-page printable calibration object
*-L/R/C.bmp	Sample stereo pairs and color files
*.ini	Sample calibration files
samples	sample client program sources
src	SVS library sources
svsclass.h	main library header (C++)
XXXgrab.cpp	framegrabber interface functions

2 Getting started with *smallv*

The *smallv* program is a standalone application that exercises the SVS library. It is a GUI interface to the stereo programs, and in addition can load and save stereo image sequences. The *smallv* program is a useful tool for initial development of a stereo application, and can also be used to check out and adjust a stereo camera setup.

The *smallv* program is in the `bin/` directory. It requires shared libraries for the stereo algorithms (*svs*), display (*fltk*), and calibration (various), all of which are in the `bin/` directory. Under MS Windows, these shared libraries (DLLs) must be in the same directory as the *smallv* program, or in the system DLL directory. Under UNIX, the `LD_LIBRARY_PATH` variable must have the path to the libraries.

Figure 2-1 shows the startup screen of the program. The black windows are for display of image and stereo data. The display programs in SVS use the FLTK cross-platform window interface, and work best in 24 bit video display mode. The version of the program is indicated in the text information area, and the title bar.

smallv will accept stereo images from either a live video source, or a stored file. The easiest way to get started with the program is to open a stored stereo sequence. From the **File** menu, choose **Open**, and navigate to the `data/` directory. The file `face320-cal-X.bmp` contains a stereo frame at 320x240 resolution. When you open it, it will show in the display windows. In the **Function** area, pull down the list box and choose **Stereo**. Finally, press the **Continuous** button to compute the stereo disparity and display it. You should see a green pattern representing stereo disparities in the right window. Under the **Horopter** label, click the **X offset** button a few times to see the effect of changing the stereo search area; a value of -4 or so should bring the close parts of the face into range. Clicking the **3D Display** button brings up an OpenGL window with a 3D view of the stereo data.

The rest of this section explains the operation of *smallv*. Since *smallv* exercises most of the functionality of the SVS libraries, it should serve as a general introduction to the SVS functions. If you are interested in using a particular framegrabber and set of cameras with *smallv*, please see Section 2.1. The

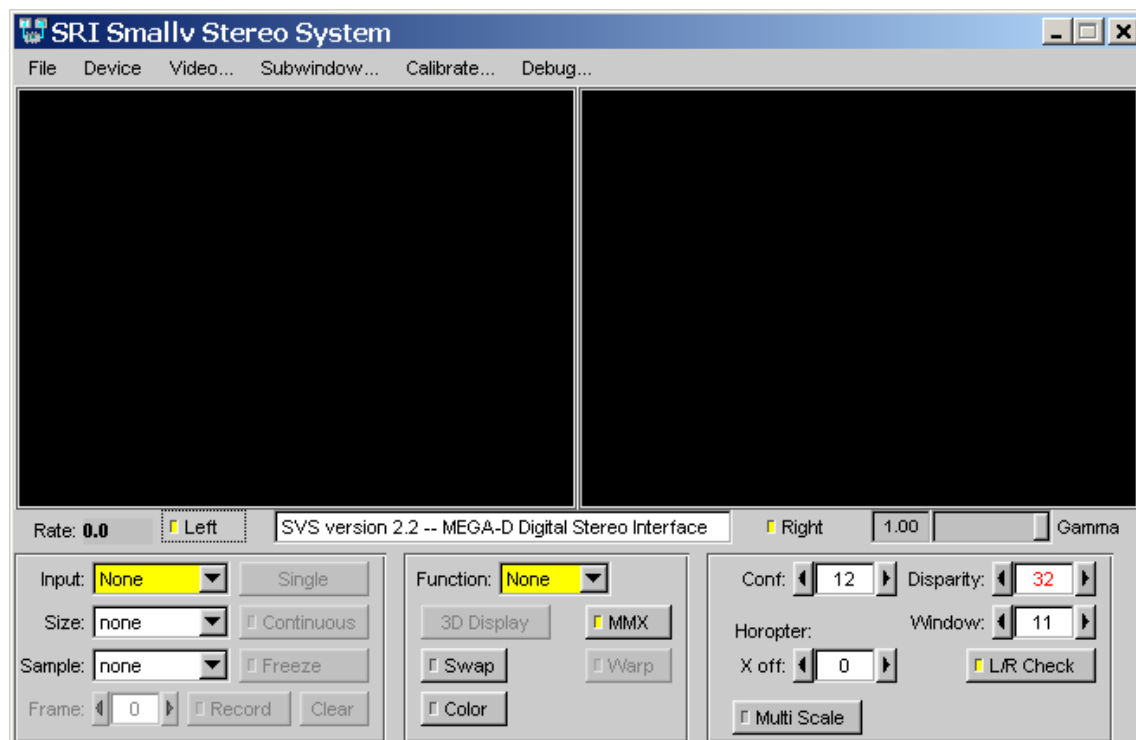


Figure 2-1 Smallv program interface. The two black windows are for display of input images and stereo results.

framegrabber interface is indicated in the message area on startup. In this case, it is the MEGA-D digital stereo head.

2.1 Inputting Live Stereo Video

The SVS libraries provide support for live video as stereo input. To input video, you must do the following steps.

1. Decide on a stereo head and framegrabber.
2. Install the framegrabber, following instructions that come with the framegrabber or the stereo head (STH-V3 or MEGA-D).
3. Copy the appropriate framegrabber DLL to `bin\svsgrab.dll` (MS Windows), or `bin/libcap.so` (Unix); see Section 2.1.2).
4. Set the appropriate video format using the Video Format menu.
5. Set the video frame size.
6. Set the input mode to video.

This section gives details necessary for performing these steps.

2.1.1 Stereo heads

Stereo requires two images from different viewpoints. The most common way to get these images is to use two identical cameras separated by a horizontal baseline. It is important the cameras have lenses with the same focal length, and that the pixel elements have the same size.

The baseline is typically from 3 to 8 inches wide, and the cameras are aligned parallel to each other,

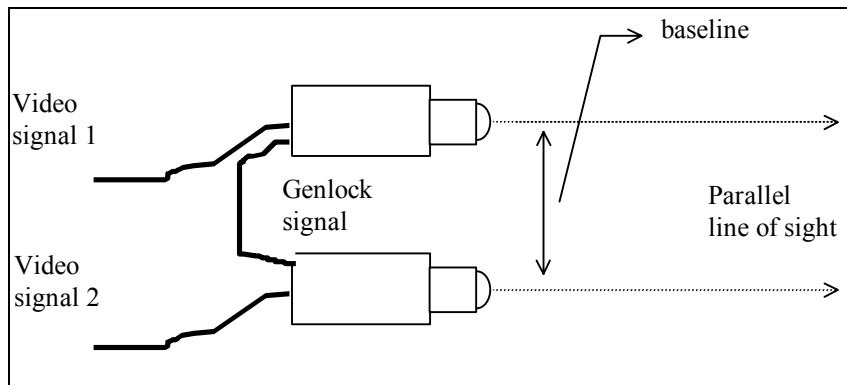


Figure 2-2 Stereo camera setup. Cameras are positioned with parallel lines of sight. Their video signals are synchronized using cross-fed signals.

although other configurations are possible. Figure 2-2 shows a typical stereo camera setup. Two cameras are pointed in the same direction, and they are connected by a cable that *genlocks* the cameras, that is, synchronizes them so that they capture images at the same time. Genlocking is important if there is any motion in the scene. If the cameras are not genlocked, they can capture the image at slightly different times, and any moving objects will be at a slightly different position in one camera relative to the other, than if it they had taken the image at the same time. If the scene is static, then genlocking is not necessary. Not every camera can be genlocked; check that the ones you have can be.

A word about monochrome vs. color cameras. If your application does not need color, it is preferable to use monochrome cameras, because stereo relies only on the luminance component of the video signal. Monochrome cameras have much better spatial resolution and dynamic range than color cameras of the same quality, since they do not have to deal with three color channels. Having said this, the MEGA-D megapixel cameras have such high resolution that using color imagers is generally not a problem, since most applications can use 640x480 or 320x240 image sizes, and the color imagers produce excellent quality by binning (averaging) a set of pixels.

The cameras produce two video outputs, which must be input to the PC running the SVS system. There are three ways to get these video signals into the PC.

1. Use two framegrabbers, and input one signal on each.

2. Use a single framegrabber capable of inputting 2 monochrome channels, e.g., the Matrox Meteor RGB or Meteor II / Multichannel.
3. Use a single framegrabber, and a stereo head that interlaces two video signals onto a single video stream. The STH-V3 from Videre Design (www.videredesign.com) is one such stereo head.
4. Use a digital stereo head, the MEGA-D (STH-MD1) from Videre Design. This stereo head outputs a digital signal on the 1394 bus, and any OHCI (Open Host Controller Interface) card can be used to input the video.

The SVS libraries can work with any size video frame up to 1288 by 1032 pixels. Standard NTSC cameras capture frames up to 640 by 480, as a set of two fields, each 640 by 240. The camera first captures a field in 1/60 of a second (the *even* field), then captures a second field 1/60 of a second later (the *odd* field). The framegrabber can put these together to form a single image of 640 by 480 size. However, the same problem with motion between non-genlocked cameras can occur on a single camera that combines fields. There is a slight time delay between fields, leading to *motion blur* in the composed frame. For this reason, the SVS libraries use fields rather than frames, so the maximum video size for NTSC signals is 640 by 240.

PAL cameras are also support by the SVS libraries, as long as the framegrabber can input PAL video. They involve similar considerations, but their maximum field size is 768 by 288.

Larger frame sizes with synchronized cameras are possible by using nonstandard *progressive scan* analog cameras, or digital cameras.

2.1.2 Analog Framegrabbers

The SVS libraries include support for a number of popular analog signal framegrabbers; for digital stereo heads, see the next subsection. The table below lists them according to their operating system.

Operating System	Framegrabber	Library	MSW Installation File
Linux	Matrox Meteor, Meteor RGB, Meteor PPB	bttvcap.so	
	Any Bt848-based card, e.g. Intel Smart Video Recorder III Imagination PXC2000	bttvcap.so	
	None	nullcap.so	
MS Windows 95/98/2000	Matrox Meteor, Meteor RGB, Meteor PPB	svsmet.dll	setup_met.bat
	Matrox Meteor II	svsmet2.dll	setup_met2.bat
	Imagination PXC200	svspxc.dll	setup_pxc.bat
	None	svsnull.dll	setup_null.bat
MS Windows NT 4.0	Matrox Meteor, Meteor RGB, Meteor PPB	svsmet.dll	setup_met.bat
	Meteor II	svsmet2.dll	setup_met2.bat
	Imagination PXC200	svspxc.dll	setup_pxc.bat

Table 2-1 Framegrabbers supported by SVS.

Under MS Windows, a particular framegrabber is accessed from the SVS libraries by copying the corresponding DLL and LIB files. Copy the appropriate C++ library (.dll) for your framegrabber to svsggrab.dll, and the corresponding reference file (.lib) to svsggrab.lib. For example, if you have installed the Matrox Meteor II board, then copy the following files in the bin\ directory:

```
svsmet2.dll -> svsggrab.dll
svsmet2.lib -> svsggrab.lib
```

There are convenient BAT files to set up a particular framegrabber, in the bin\ directory. Simply double-click on the batch file to perform the required copying. For example, to set up the Imagination PXC200 framegrabber, go to the bin\ directory in Windows Explorer, and double-click on the file setup_pxc.bat.

The framegrabber interface must be set up before starting `smallv`; it cannot be changed while the program is running.

Under Linux, the interface to either the Matrox Meteor cards or a Bt848 card (PXC200, Intel Smart Video Recorder, etc.) is with the shared library `bttvcap.so`. Copy this library to `bin/libcap.so` to use it. You must load the proper low-level driver for the card, namely, the BTTV drivers. These drivers are included with almost every distribution of Linux; we recommend the current Red Hat distribution.

The framegrabber interface used by `smallv` under MS Windows or Linux is indicated in the message window at startup.

There are some limitations in framegrabber drivers that should be noted. First, there are currently no good analog framegrabbers for portables. For the digital interface, any standard IEEE 1394 OHCI card can be used; this is one of the advantages of the digital interface.

2.1.3 IEEE 1394 (FireWire) Framegrabber

The SVS has an interface to digital stereo heads from Videre Design via the IEEE 1394 serial bus. Any OHCI-compliant IEEE 1394 PCI or PCMCIA card can be used, under MS Windows 98/2000 or Linux. Please check the stereo head manual for instructions on installing the 1394 card and drivers.

The relevant interface libraries are given in Table 2-2 below. To set up a particular interface in Linux, copy the library file to `bin/libcap.so`. Under MS Windows, execute the setup file in the `bin\` directory by double-clicking on it in Windows Explorer.

Operating System	Stereo Head	Library	MSW Installation File
Linux	MEGA-D	<code>libpix.so</code>	
	Dual DCAM	<code>libdcam.so</code>	
MS Windows 98/ME/2000	MEGA-D	<code>svspix.dll</code>	<code>setup_megad.bat</code>
	Dual DCAM	<code>svsdcam.dll</code>	<code>setup_dcam.bat</code>

Table 2-2 Interface libraries for Videre Design digital stereo heads.

2.1.4 Selecting Devices

The *Device* menu button lets you tell the SVS library what kind of video input you are using. For Videre Design digital heads (MEGA-D and Dual DCAM), you can select among multiple devices attached to any IEEE 1394 card on your computer. For analog stereo heads, you can choose how to input images from the analog framegrabber boards. Currently, there is no way to select multiple analog stereo heads connected to the same computer.

Digital Stereo Devices (MEGA-D and Dual DCAM)

A digital stereo head attached to the IEEE 1394 bus is recognized by `smallv`, and the *Device* menu button will drop down a list of these devices when selected. Devices have a *number*, which starts at 1 for the first device encountered. These numbers can change as devices are added or dropped from the bus. Devices also have an *id*, which is a numeric string that is unique to the device. The *Device* list shows both the current device number, and its unique id. The currently selected device is indicated by a checked box; you can change the current device by selecting any available device. This choice becomes active the next time *Video* input is selected in the input choice box.

Only one type of device, the MEGA-D or Dual DCAMs, will be seen by the `smallv` program. The choice depends on which interface library has been loaded (see Section 2.1.3). It is possible to mix these devices on the same IEEE 1394 bus, but a given application will see only one type of device or the other.

Analog Stereo Devices (STH-V3 and user cameras)

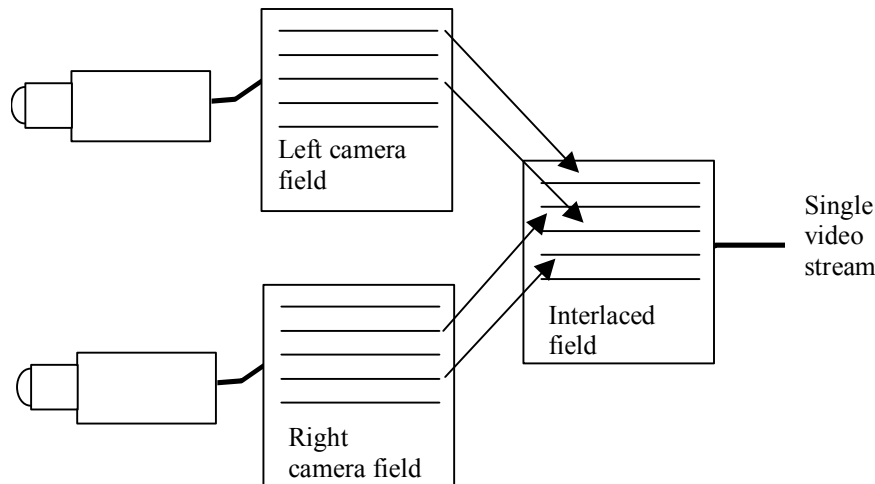


Figure 2-3 Line-interlaced stereo cameras (STH-V3 stereo head). Alternate lines from each camera are interlaced into a single video stream.

Analog camera input goes through an analog framegrabber, where it is converted into digital form and sent to the host computer memory. The `smallv` interface will input video from the cameras through either one or two framegrabbers, depending on the type of setup. There are three choices:

1. Line interlace [default]. This is the mode for the STH-V3 line interlace stereo head. Any framegrabber can be used in this mode.
2. Dual framegrabbers. This mode uses two framegrabbers, with one framegrabber per camera. Check Table 2-1 for supported framegrabbers.
3. RG components. This mode uses the Matrox Meteor RGB or Meteor II, and inputs one camera video stream on the R channel, and one on the G channel.

It is important for stereo processing to have the left camera image appear as the left image in the `smallv` program. Once the video input is displayed, you can check this by pointing the cameras along your line of sight. The right camera appears on your right side, and the right image on the `smallv` display should show this image. You can cover one camera with your hand, and observe which displayed image goes dark. With dual framegrabbers or RG input, the solution to having the wrong camera inputs is to simply switch the inputs, or to use the `swap` button in `smallv`, which interchanges the images in memory.

Under line interlace mode (Figure 2-3), the first horizontal line of a video field is from the left camera, the second from the right, the third from the left, and so on, making a single video stream. The SVS software de-interlaces the video stream, reconstructing the left and right images in memory, at half the original vertical resolution. Because of the variation in how framegrabbers determine which is the first line of a field, the SVS software will sometimes switch the left and right fields during de-interlacing. The `swap` button switches the left and right fields during deinterlacing.

The `smallv` application interfaces only to one analog stereo head, using the first one or two framegrabbers that it finds. It is possible to specify other framegrabbers from a use application, using the SVS API.

2.1.5 Frame Size

The SVS libraries as delivered can work with frame sizes up to 1288 by 1032. In fact, the SVS algorithms can work with arbitrarily sized frames.

A subset of frame sizes are supported for video input in the `smallv` application; the following table summarizes them. Most framegrabbers support hardware interpolation and scaling, so that bus traffic is minimized by working with smaller frames. The exception is the Meteor RGB, which passes a full field to memory, where it is decimated by the SVS software.

Video Format	Frame Sizes
Line interlace	160x120
	320x120
Dual framegrabber and RG component	640x240
	320x240
	320x120
	160x120
1394 (digital) interface	1280x960, 640x480, all others above

**Table 2-3 Frame sizes available for video input
in smallv.**

Video frame size is selected with the *Size* drop list in the *Source* area. Video size can be changed only when frames are not being acquired. Once acquisition starts, the frame size is fixed.

2.1.6 Image Sampling

The sampling for analog framegrabbers is implicit in the frame size. For example, if the camera image size is 320x240, and the requested frame size is 160x120, then the full image is scaled down by the framegrabber, usually using interpolation to produce a smooth image.

With the MEGA-D digital interface, the user has full control over the sampling method, and the *Sample* and *Size* controls combine to produce the final result. For example, if the sampling mode is x1 (no subsampling), then an image size of 320x240 produces a *subwindow* within the full image. (Subwindowing is not available from analog framegrabbers supported by SVS.) The placement of the subwindow can be changed in real time under program control, using the dialog from the *Subwindow...* menu.

There are several sampling modes. *Decimation* samples the image by removing pixels, e.g., “x2 dec” means that every second pixel in a line is removed, and every other line is removed. *Binning* samples the image by averaging over a block of four pixels, to produce the same result. Binning produces smoother images with less noise, but it is slower than decimation, which is done by the stereo hardware. Combination sampling modes are available, e.g., “x4 bin+dec” samples the image down to 1/4 size in horizontal and vertical directions, by decimating by 2 and then binning by 2.

The Dual DCAM stereo device supports a single sampling mode, x2 binning at 320x240 output resolution. The binning mode will reduce video noise for this device.

2.1.7 Image Source

The source for stereo images can be either a memory buffer or a live video stream. The *Source* drop list lets you choose between these, or to stop any input. Buffer input is discussed in Section 2.1.9.

2.1.8 Streaming Mode

Images from video cameras or the buffer can be processed in three acquisition modes. Only one acquisition mode is active at a given time.

- Continuous mode. In this mode, stereo pairs are continuously input, processed, and displayed. The maximum frame rate is 30 Hz for live analog image data, and up to 80 Hz for the MEGA-D digital system. See Section 3.7 for performance information. The rate is indicated next to the text information area.
- Single frame mode. In this mode, a single stereo pair is input, processed, and displayed each time the *Single* button is pressed.
- Freeze mode. In this mode, a single stereo pair is input, then the same frame is continuously processed and displayed. This mode is useful in checking the effect of different stereo parameters on the same image.

2.1.9 Adjusting Video Parameters

Most framegrabbers support some kind of video image adjustments, such as contrast or brightness. The video parameter dialog is invoked using the Video... menu item (Figure 2-4).

Most analog cameras have automatic adjustment of exposure and gain, which change according to lighting conditions. The user can set brightness and contrast, which are framegrabber parameters that change the processing of the analog signal.

The MEGA-D digital stereo head has manually controlled exposure and gain. Exposure is the time that any given pixel is exposed to light before being read out. Gain is a amplification of the signal that comes out of the pixel. In general, it is best to increase the exposure first, and if necessary, to increase gain once exposure reaches a maximum. The reason for this is that gain will increase the video noise, while exposure increases the pixel's response to light. In some cases, though, short exposure times are desirable for minimizing motion blur, and it may be more convenient to increase gain while exposure is not at a maximum.

The values of exposure, gain, brightness, and contrast are all represented as a percent.

The colorized version of the MEGA-D digital camera can input color images, and the color balance can be adjusted manually using the red/blue differential gain. More information about color processing is in Section 2.1.12.

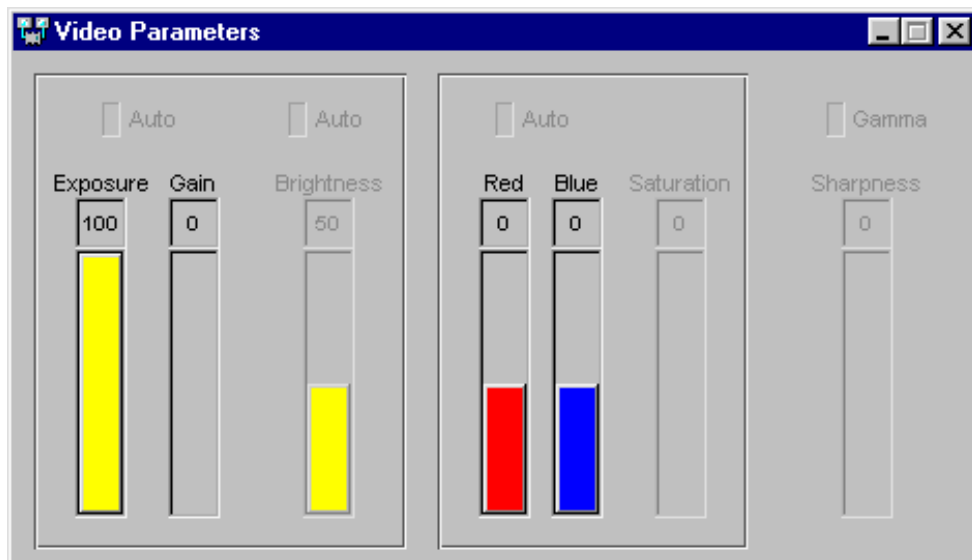


Figure 2-4 Video Parameter dialog box.

2.1.10 Subwindowing

The MEGA-D digital stereo head can send to the host computer just a portion, or *subwindow*, of the stereo image. For example, if the MEGA-D is in x2 sampling mode (full-size image is 640x320), and the image size is chosen to be 320x240, then `smallv` will input only a 320x240 subwindow of the full image. Figure 2-5 shows two of these subwindows, and the original full-size image.

The placement of subwindows is controlled by the vertical (Y) and horizontal (X) offset controls in the Subwindow dialog window; the dialog is initiated from the Subwindow... menu item in the main window. These parameters can be changed in real time, enabling electronic panning of the live image.



Figure 2-5 Two 320x240 subwindows (bottom) of a 640x480 image (top).

2.1.11 Vergence

When in subwindow mode, the two cameras in a stereo rig generally will have the same X and Y offsets, so that they keep the parallel line-of-sight characteristic of the stereo rig. However, for viewing close objects, it is advantageous to toe-in, or *verge*, the two stereo cameras. In this way, the images of the near object will both contain the object in the center.

Human eyes verge mechanically when viewing close objects. Mechanical vergence for stereo cameras is difficult, however, since it involves complicated motor control, and more importantly, disturbs the calibration that is critical for stereo analysis. Instead, with the subwindow capability of the MEGA-D, it is possible to verge the stereo images electronically, by choosing appropriate horizontal offsets for each image.

Figure 2-6 shows the effects of using electronic vergence. The top stereo pair, of a close object, puts the object into the center of the left frame. In the right frame, the object has a large disparity and is visible in the left side of the frame.

The bottom stereo pair is created by adding vergence to the subwindow process, offsetting the right subwindow horizontally by 120 pixels, relative to the left subwindow. Both frames now have the near object centered.

Vergence of the subwindows is set using the vergence control in the Subimage box of the Subwindow dialog. It is a real time control, just like the X and Y subwindow offsets.



Figure 2-6 Parallel image subwindows (top) and verged image subwindows (bottom), showing a close object.

2.1.12 Color

As of Version 2.1, SVS supports color input and display. Besides the two monochrome left/right stereo channels, there is a third color channel that corresponds to the left image, with images in RGB 32-bit format, and optionally a fourth color channel for the right image. The color channels do not participate in stereo processing, but can be useful in applications that combine color and stereo information, for example, object tracking. Usually only the left color channel is needed, since the left image is the reference image for stereo disparities and 3D information.

Color information from the MEGA-D digital head (STH-MD1-C) is input as raw colorized pixels, and converted by the interface library into two monochrome and one or two RGB color channels. The main color channel corresponds to the left image, which is the reference image for stereo. The color image can be de-warped, just like the monochrome image, to take into account lens distortion (Section 4). Optionally, a second color channel is available for the right image.

Color information from the camera is input only if the `Color` button is pressed on the main window (Figure 2-1). To get the right color image, use the `SetColor()` command from an application.

Because the typical color camera uses a colorizing filter on top of its pixels, the color information is sampled at a lower resolution than a similar non-colorized camera samples monochrome information. In general, a color camera has about $\frac{1}{4}$ the spatial resolution of a similar monochrome camera. To compensate for the reduced resolution, use binning (Section 2.1.6) to increase the fidelity of the image. For example, if you need a 320x240 frame size, use 640x480 and binning x2.

The relative amounts of the three colors, red/green/blue, affects the appearance of the color image. Many color CCD imagers have attached processors that automatically balance the offsets among these colors, to produce an image that is overall neutral (called *white balance*). The MEGA-D provides manual color balance by allowing variable gain on the red and blue pixels, relative to the green pixels. Manual balance is useful in many machine vision applications, because automatic white balance continuously changes the relative amount of color in the image.

The manual gain on red and blue pixels is adjusted using the `Video Parameters` window (Section 2.1.9). For a particular lighting source, try adjusting the gains until a white area in the scene looks white, without any color bias.

2.2 Storing, Saving, and Loading Stereo Data

`smallv` provides a basic facility for loading and saving stereo data streams. The file load and store functions are part of the SVS library, and their source code is included. `smallv` exercises these functions, and provides a memory buffer for storing live stereo video. In `smallv`, the buffer always holds the left and right input stereo images.

2.2.1 Stereo Stream Storage

`smallv` has an internal buffer capable of holding 30 stereo pairs (frames) at a 640x480 frame size. The buffer will hold more frames at smaller sizes, fewer frames at larger sizes. The buffer can be filled from a previously-saved file, or from live video input. The buffer can also be written out to a file, and used as the source for stereo processing in `smallv`. The current frame is indicated in input information area.

When the input source is the buffer, the acquisition mode controls (`Continuous`, `Single`, `Freeze`) control the processing of the buffer frames (Section 2.1.8). The frame control can also be used to go to an individual frame when in `Single` acquisition mode.

The `Record` button controls the input of live video into the buffer. `Clear` clears the buffer and resets it to frame 0. Activating the `Record` button starts the input of live video frames into the buffer. The source must be set to `Video`; either `Continuous` or `Single` mode may be used. Frames are stored sequentially until the buffer is full. Pressing `Record` again will also turn off acquisition..

As an example, to capture a short video sequence and replay it, perform the following steps.

1. Start acquiring live video in continuous mode.
2. Clear the buffer (`Clear` button).
3. Start buffer storage (`Record` button).
4. After a short period, stop buffer storage (`Record` button).
5. Change from `Video` to `Buffer` source.

At this point, the short segment that is in the buffer will be replayed as a short continuous loop. The buffer, or individual images, can be saved to a file.

2.2.2 Loading and Storing Files

The SVS libraries work with different file types for image storage.

- **BMP format.** Each BMP file contains a single 8-bit grayscale image, or an RGB 24-bit color image. The color coding for the 8-bit BMP file is 256 shades of gray, with 0 being black and 255 white. By convention, a stereo pair is saved as two files with the linked names `XXX-L.BMP` (left image) and `XXX-R.BMP` (right image). The corresponding color file is saved as `XXX-C.BMP`. Finally, the image parameters are stored as a text file `XXX.INI`.
- **PNG format.** PNG (Portable Network Graphics) allows for storage of non-standard formats, such as 16-bit grayscale. This format is used to store disparity images, which are grayscale images with pixels larger than 8 bits.
- **Text files for disparity images.** Disparity images can be saved as a text file, with one line of text for each line of the image (e.g., a 320x240 image will have 240 lines). Each line contains an image row of disparities, as integers. The special values `-1` and `-2` indicate that the disparities were filtered out, by the texture measure (`-1`) or the left/right check (`-2`).
- **Text files for 3D points.** 3D point arrays, generated from a disparity image (Section 2.4.2), can be saved as a text file. Each line of the file represents one point of the array. The array has the same format as the image from which it was produced, e.g., if the input image is 320x240, then the file has 320x240 lines, in row-primary order. Each line has 3D X,Y,Z coordinates first, as floating-point numbers, then three integers for the R,G,B values of the pixel at that point. If the disparity at a pixel is filtered out, then the Z value is negative, indicating a filtered value.

Images and image sequences are loaded into and stored from the buffer using the `File` menu. To load stereo frames, use the `Load` menu item to bring up a file choice dialog. Choosing either BMP file of a pair automatically loads the other. In addition, if a color file or parameter file (`.ini`) is present, it is also loaded.

To save the buffer to a file, use the `Store Buffer` menu item. This saves the buffer as a set of BMP files, appending a decimal number to each, starting at 001. Alternatively, to save the current frame as two BMP files, use the `Store Current` menu item. If stereo processing is active, then the stereo disparity image is saved also as a PNG file; this is the only method to save a disparity image from the `smallv` application. Color information, if present, is saved as a 24-bit BMP file.

2.3 Display

`smallv` displays two images in its display area. The left display is always the left input image. Input images are displayed in grayscale, unless color information is present: in this case, the left image will be shown in color.

The right display can be either the right input image, or the results of stereo processing. Processing results are always displayed in “greenscale”, using shades of green.

Either display can be turned off by unchecking the box underneath the display area. Turning off the display will let `smallv` run faster.

Images larger than 320x240 are automatically scaled down by factors of 2^n to fit into a 320x240 area. Smaller image sizes are displayed in the original size.

To display properly for human viewing, most video images are formatted to have a nonlinear relationship between the intensity of light at a pixel and the value of the video signal. The nonlinear function compensates for loss of definition in low light areas. Typically the function is x^γ , where γ is 0.45, and the signal is called “gamma corrected.” Digital cameras, such as the MEGA-D, do not necessarily have gamma correction. This is not a problem for stereo processing, but does cause the display to look very dark in low-light areas. You can add gamma correction to the displayed image by choosing an appropriate gamma value in the slider under the right display window (Figure 2-1).

2.4 Stereo Processing and Parameters

In `smallv`, stereo processing takes place in conjunction with the input of stereo images. The basic cycle is:

get stereo pair -> process pair -> display pair

The input is either from live video or the buffer (Sections 2.1 and 2.1.9). In freeze mode, the same pair is processed continuously, so adjustments can be made in stereo parameters.

2.4.1 Stereo Function

Stereo processing is turned on by choosing `Stereo` from the `Function` drop list. The stereo disparity image will appear in the right display. Stereo disparities are encoded by green: brighter green is a higher disparity, and therefore closer to the cameras (see Section 2.4.4 for a technical description of disparity).

Disparities represent the distance between the horizontal appearance of an object in the stereo images. The stereo process interpolates this distance to 1/16 pixel, e.g., a disparity value of 45 represents a displacement of $2\frac{13}{16}$ pixels. The maximum displacement currently supported is 80 pixels, so disparity values range from 0 (no disparity) to 1280. Disparity values are returned as 16-bit (short) integers. The values 0xFFFF and 0xFFFE are reserved for filtering results (Section 2.5)

If `smallv` is running on an MMX processor (Pentium or AMD) then stereo processing is much faster, taking advantage of the parallel data operations. The processor is queried and the MMX box is checked if the instructions are available. You can turn the MMX processing on and off by toggling the box. But, if your system does not have MMX instructions, you will not be able to turn it on.

2.4.2 3D Transformation

A pixel in the disparity image represents range to an object. This range, together with the position of the pixel in the image, determines the 3D position of the object relative to the stereo rig. SVS contains a function to convert disparity values to 3D points. These points can then be displayed in a 3D viewer.

To take the current disparity image and display it in 3D, press the `3D Display` button. An OpenGL window will show the 3D points constructed from the disparity image, and you can change the viewpoint of the window to see the 3D structure (Figure 2-7).

The coordinate system for the 3D image is taken from the optic center of the left camera of the stereo rig. Z is along the optic axis, with positive Z in front of the camera. X is along the camera scan lines, positive values to the right when looking along the Z axis. Y is vertical, perpendicular to the scan lines, with positive values down.

The X and Y position of the viewpoint, as well as rotation around the Z axis, can be changed with the sliders on the left side of the window. The scale of the image can be changed as well. Finally, the viewpoint can be rotated around a point in the image, to allow good assessment of the 3D quality of the stereo processing. The rotation point is selected automatically by finding the point closest to the left camera, near the optic ray of that camera. To rotate the image around this point, put the mouse in the 3D window, and drag the pointer while holding the left button down.

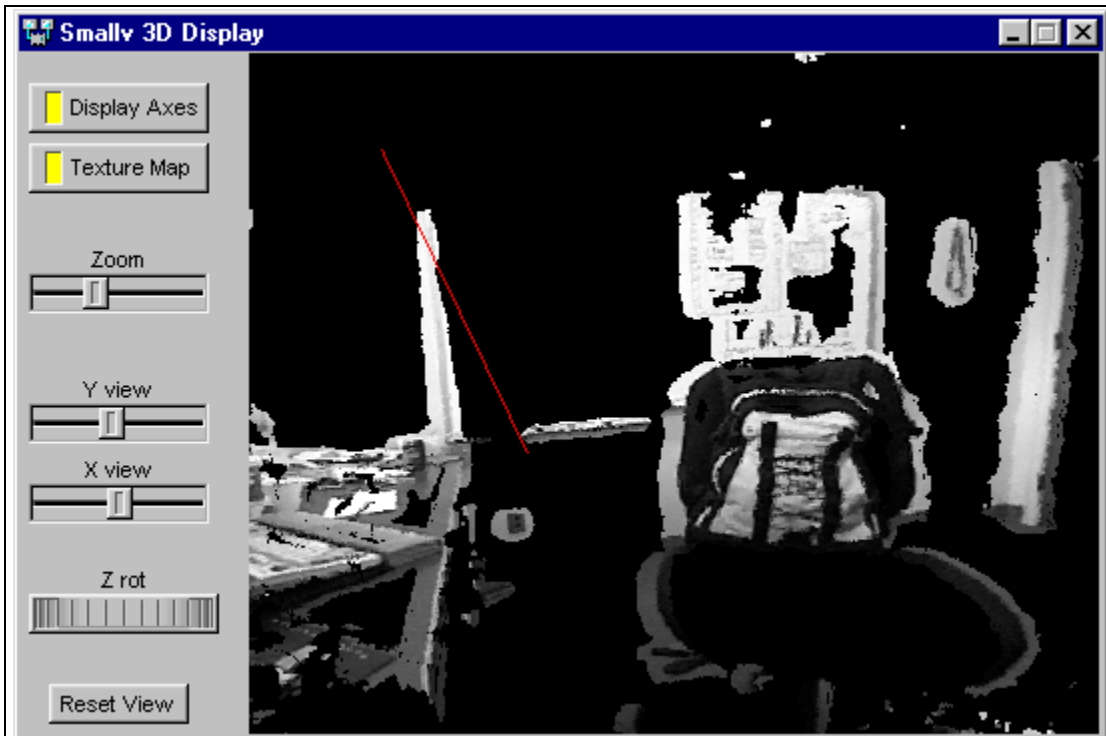


Figure 2-7 3D display window. The red ray is the optic ray from the left camera.

2.4.3 Calibration

For good stereo processing, the two images must be aligned correctly with respect to each other. The process of aligning images is called *calibration*. Generally speaking, there are two parts to calibration: *internal calibration*, dealing with the properties of the individual cameras and especially lens distortion; and *external calibration*, the spatial relationship of the cameras to each other. Both internal and external calibration are performed by an automatic calibration procedure described in Section 4. The procedure needs to be performed when lenses are changed, or the cameras are moved with respect to each other.

From the internal and external parameters, the calibration procedure computes an image warp for rectifying the left and right images. In stereo rectification, the images are effectively rotated about their centers of projection to establish the ideal stereo setup: two cameras with parallel optical axes and horizontal epipolar lines (see Fig. 2-2). Having the epipolar lines horizontal is crucial for correspondence finding in stereo, as stereo looks for matches along horizontal scanlines.

Figure 2-8 shows a pair of images of the calibration target taken with the MEGA-D stereo head and a 4.8 mm wide-angle lens. In the original images on the top, there is lens distortion, especially at the edges of the image: notice the curve in the target. Also, the images are not aligned vertically.

The bottom pair is the result of calibrating the stereo head and then rectifying the two original images. Now the images are aligned vertically, and all scene lines are straight in the images.

Figure 2-9 shows sample disparity images for uncalibrated and calibrated cameras. Without calibration, it is impossible for the stereo algorithms to find good matches.

2.4.4 Disparity Search Range

Even with stereo rectification, it may not be possible to match every object in the scene, because the horopter is not large enough. In this case, the horopter can be enlarged by changing the number of disparities searched by the stereo process. This search range can vary from 8 to 80 pixels. Larger search ranges enlarge the horopter, but not in a linear fashion, i.e., a search range of 32 does *not* give twice the

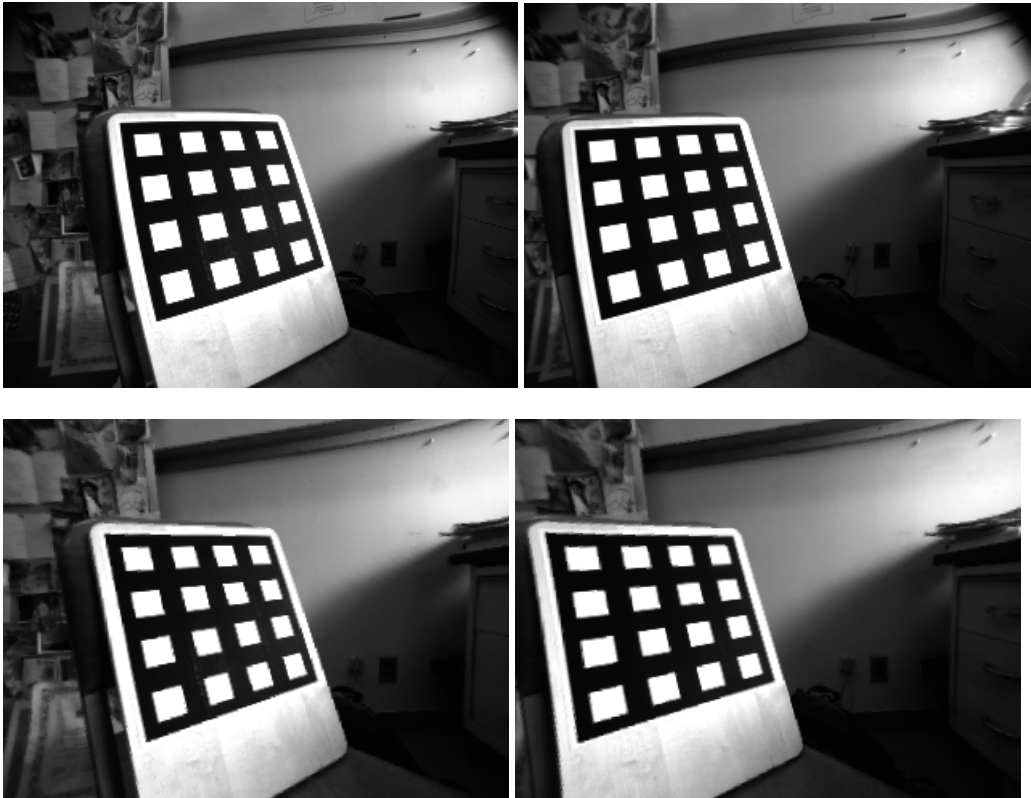


Figure 2-8 Original stereo pair (top) and rectified pair (bottom).

horopter range of 16; see Section 4 for technical details.

Changing the disparity search size affects the time it takes to process stereo. A search space of 32 pixels will take about twice as long as a search space of 16 pixels. It will actually take a little less, because there is some fixed overhead in processing the images. Obviously, the smallest search range necessary for the application is the best choice.

Disparities are interpolated to 1/16 pixel, so a search range of 16 means that there are 256 integral disparity values, ranging from 0 (no disparity) to 255 (maximum disparity of 15 15/16 pixels).

The search range is selected using the `Disparities` value in the `Parameters` area. When the range is switched, the disparity image will lighten or darken to reflect the changed values of disparities.

2.4.5 Adjusting the Horopter

The stereo rectification procedure sets up the horopter, or depth of field of stereo, so that objects are matched from infinity to some distance in front of the camera. Objects closer than this near point will not be matched, and will produce random disparity readings. The near point distance is a function of the search size, the stereo baseline, and the focal length of the camera lenses. One can adjust the horopter by adjusting a horizontal X offset, moving the depth range closer to the camera. The depth range desired in the end application would drive the setting of this parameter. For example, if the image does not contain any objects farther than a certain distance, the X offset can be adjusted so that the far point of the horopter is at that distance. Changing the X offset causes the disparity display to get uniformly lighter or darker, as the horopter is shifted and the disparity of an object changes. Adjusting the horopter to cover a specific range of depths is discussed in Section 4.

2.4.6 Pixel Information

SVS will show pixel information when the left button is clicked in either SVS display window. The information is displayed in the text window in the format:

```
x232 y120 [131] [11] Xaaa Ybbb Zccc
```

The image coordinates of the mouse are given by the `x`, `y` values. The values in square brackets are the pixel values of the left and right images. If the right image is displaying stereo disparities, then the right value is the disparity value. Finally, the `X`, `Y`, `Z` values are the real-world coordinates of the image point, in mm. Note that `X`, `Y`, `Z` values are calculated only if stereo is being computed, and to be accurate, a good calibration file must be input (Section 4).

2.4.7 Correlation Window Size

The size of the correlation window used for matching affects the results of the stereo processing. A larger window will produce smoother disparity images, but will tend to “smear” objects, and will miss smaller objects. A smaller window will give more spatial detail, but will tend to be noisy. Typical sizes

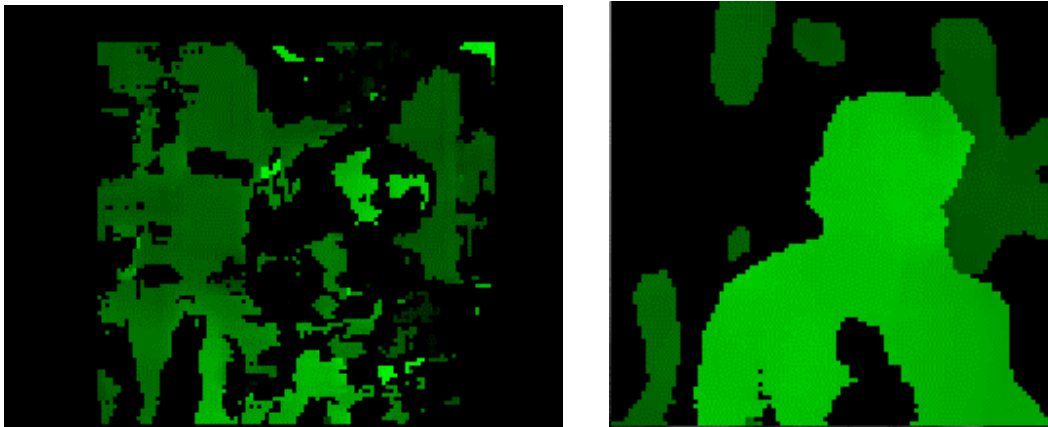


Figure 2-9 Uncalibrated (left) and calibrated (right) disparity images.

for the window are 9x9 or 11x11. The window size is selected using the `Sum window` drop list. In the MMX implementation, not all window sizes are supported. More technical information on the correlation window can be found in Section 3.4.

2.4.8 Multiscale Disparity

Multiscale processing can increase the amount of information available in the disparity image, at a nominal cost in processing time. In multiscale processing, the disparity calculation is carried out at the original resolution, and also on images reduced by 1/2. The extra disparity information is used to fill in dropouts in the original disparity calculation (Figure 3-8 in Section 2.4.8).

Multiscale processing is turned on in `smallv` by enabling the `MultiScale` button.

2.5 Filtering

Stereo processing will generally contain incorrect matches. There are two major sources for these errors: lack of sufficient image texture for a good match, and ambiguity in matching when the correlation window straddles a depth boundary in the image. The SVS stereo processing has two filters to identify these mismatches: a confidence measure for textureless areas, and a left/right check for depth boundaries.

Areas that are filtered appear black in the displayed disparity image. To distinguish them from valid disparity values, they have the special values 0xFFFF (confidence rejection) and 0xFFFE (left/right rejection).

2.5.1 Confidence Filter

The confidence filter eliminates stereo matches that have a low probability of success because of lack of image texture. There is a threshold, the confidence threshold, that acts as a cutoff. Weak textures give a confidence measure below the threshold, and are eliminated by the filter.

The confidence threshold is adjusted using the `Conf` spin control in the `Parameters` area. A good value can be found by pointing the stereo cameras at a textureless surface such as a blank wall, and starting the stereo process. There will be a lot of noise in the disparity display if the confidence threshold is set to 0. Adjust the threshold until the noise just disappears, and is replaced by a black area.

The computational cost of the confidence filter is negligible, and it is usually active in a stereo application.

2.5.2 Left/Right Filter

Each stereo camera has a slightly different view of the scene, and at the boundaries of an object there will be an area that can be viewed by one camera but not the other. Such occluded areas cause problems for stereo matches. Fortunately, they can be detected by a consistency check in which matching is done first by using the left image as a fixed base, and then repeating the match using the right image as the base. Disparity values for the same point that are not the same fail the left/right check. Typically, this will occur near the boundaries of objects.

The left/right check is controlled by three radio buttons in the `Parameter` area. It can be turned on or off. A third option is to perform the check, but instead of discarding disparity values that are inconsistent, use the one that is smaller (further away). This option can fill in the areas around object borders in a reasonable way. It is not currently available under MMX processing.

The left/right check adds about 20% to the computational cost of the stereo process, but is usually worth the effort.

2.6 Saving and Restoring Parameters

All of the parameters that control the operation of the SVS Stereo Engine can be saved to a file for later use. Parameter files can be loaded and saved using the File menu: Load Param File and Store Param File.

The file `data/megad-75.ini` contains a sample file for a 7.5 mm lens on the MEGA-D stereo rig. It serves as an example of the settings available through parameter files. In practice, these settings are usually computed using the calibration program, and then saved to a file for later use. But, it is also possible to change the settings directly in the file.

```
# SVS Engine v 2.2 Stereo Camera Parameter File

[image]                                # image frame parameters
max_linelen 1280                       # max size of imager
max_lines 960
max_decimation 4                       # allowable decimation at imager
max_binning 2                         # allowable binning in driver
gamma 0.700000                        # gamma correction for display
color 0                               # 0 for monochrome, 1 for color
ix 0                                   # subwindow offset
iy 0
vergence 0                            # vergence of right subwindow
rectified 0
width 320                             # subwindow size
height 240
linelen 320                           # window size
lines 240
decimation 2                          # current decimation and binning
binning 2
subwindow 1                          # 1 for subwindow capability
have_rect 1                          # 1 if we have rectification parameters
autogain 0                            # 1 if autogain available
manualgain 1                          # 1 if manual gain available
autowhite 0                           # 1 if auto white balance available
manualwhite 1                        # 1 if manual white balance available
gain 0                               # current gain value [0,100], neg for auto
exposure 100                         # current exposure [0,100], neg for auto
contrast 0                           # current contrast [0,100]
brightness 50                        # current brightness [0,100]
saturation 20                        # current saturation [0,100]
red 0                                # current red gain [-40,40], neg for auto
blue 0                               # current blue gain [-40,40]

[stereo]                              # stereo processing parameters
convx 9                              # prefilter kernel size
convy 9
corrsize 11                          # correlation window size
corrsize 11
thresh 20                            # confidence threshold value
lr 1                                 # left/right filter on (1) or off (0)
ndisp 24                             # number of disparities to search
dpp 16                               # subpixel interpolation
offx 0                               # horopter offset
offy 0                               # vertical image offset, not used

[external]
```

```

Tx -89.458214      # translation between left and right cameras
Ty -0.277252
Tz -0.923279
Rx -0.008051      # rotation between left and right cameras
Ry -0.003771
Rz -0.000458

[left camera]
pwidth 1280        # number of pixels in the camera
pheight 960
dpx 0.007500       # pixel spacing, mm
dpy 0.007500
sx 1.000000        # aspect ratio
Cx 582.260123      # camera center, pixels
Cy 506.081223
f 7.798704         # focal length, mm
kappa1 0.002983    # radial distortion parameters
kappa2 -0.000040
proj               # projection matrix: from left camera 3D coords
                  # to left rectified coordinates
1.041674e+003 6.177793e+000 5.666963e+002 0.000000e+000
-6.957139e+000 1.042596e+003 5.022628e+002 0.000000e+000
-6.576823e-003 -3.900478e-005 1.000000e+000 0.000000e+000
rect              # rectification matrix for left camera
1.001883e+000 -5.935817e-003 1.956143e+001
6.693463e-003 1.000873e+000 1.397722e+000
6.313708e-006 1.844243e-015 1.000000e+000

[right camera]
pwidth 1280        # number of pixels in the camera
pheight 960
dpx 0.007500       # pixel spacing, mm
dpy 0.007500
sx 1.000000        # aspect ratio
Cx 548.992956      # camera center, pixels
Cy 495.924832
f 7.834438         # focal length, mm
kappa1 0.002722    # radial distortion parameters
kappa2 -0.000021
proj               # projection matrix: from right camera 3D coords
                  # to right rectified coordinates
1.041674e+003 6.177795e+000 5.666964e+002 -9.352453e+004
-6.957140e+000 1.042596e+003 5.022628e+002 4.191134e-004
-6.576824e-003 -3.900531e-005 1.000000e+000 -9.225858e-005
rect              # rectification matrix for right camera
1.006349e+000 -9.720995e-003 -1.942293e+001
8.006440e-003 9.997475e-001 -1.474720e+000
9.887915e-006 -7.757015e-006 1.000000e+000

```

3 Stereo Geometry

Stereo algorithms compute range information to objects by using triangulation. Two images at different viewpoints see the object at different positions: the image difference is called *disparity*. This section discusses the basic equations that govern the relationship between disparity and range.

3.1 Disparity

The figure below displays stereo geometry. Two images of the same object are taken from different viewpoints. The distance between the viewpoints is called the *baseline* (**b**). The focal length of the lenses is **f**. The horizontal distance from the image center to the object image is **dl** for the left image, and **dr** for the right image.

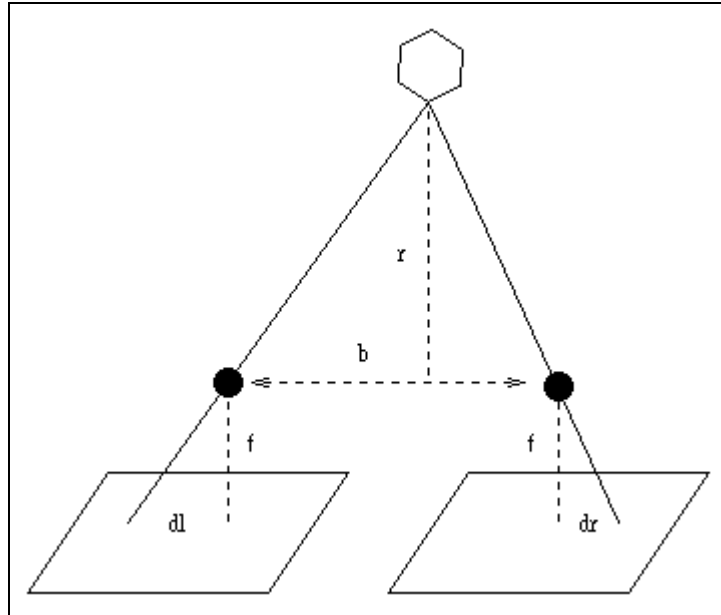


Figure 3-1. Definition of disparity: offset of the image location of an object.

Normally, we set up the stereo cameras so that their image planes are embedded within the same plane. Under this condition, the difference between **dl** and **dr** is called the *disparity*, and is directly related to the distance **r** of the object normal to the image plane. The relationship is:

$$(1) \ r = bf / d, \text{ where } d = dl - dr.$$

Using Equation 1, we can plot range as a function of disparity for the STH-V1 stereo head. At their smallest baseline, the cameras are about 8 cm apart. The pixels are 14 μ m wide, and the standard lenses have a focal length of 6.3 mm. For this example, we get the plot in Figure 3-2. The minimum range in this plot is 1/2 meter; at this point, the disparity is over 70 pixels; the maximum range is about 35 meters. Because of the inverse relationship, most of the change in disparity takes place in the first several meters.

The range calculation of Equation (1) assumes that the cameras are perfectly aligned, with parallel image planes. In practice this is often not the case, and the disparity returned by the Stereo Engine will be offset from the ideal disparity by some amount X_0 . The offset is explained in the section below on the horopter, and in the section on calibration.

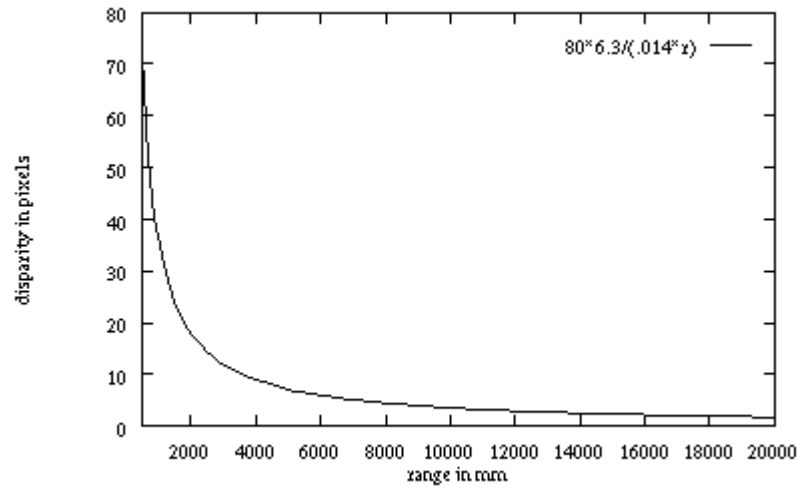


Figure 3-2. Inverse relationship between disparity and range. This plot is for a focal length of 6.3 mm, a baseline of 80 mm, and a pixel width of 14 mm.

3.2 Horopter

Stereo algorithms typically search only a window of disparities, e.g., 16 or 32 disparities. In this case, the range of objects that they can successfully determine is restricted to some interval. The *horopter* is the 3D volume that is covered by the search range of the stereo algorithm. The horopter depends on the camera parameters and stereo baseline, the disparity search range, and the X offset. Figure 3-3 shows a typical horopter. The stereo algorithm searches a 16-pixel range of disparities to find a match. An object

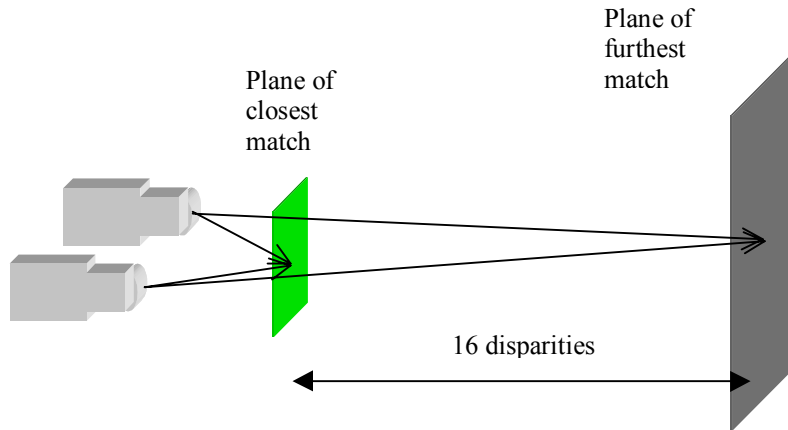


Figure 3-3 Horopter planes for a 16-pixel disparity search.

that has a valid match must lie in the region between the two planes shown in the figure. The nearer plane has the highest disparity (15), and the furthest plane has the lowest disparity (0).

The placement of the horopter can be varied by changing the X offset between the two images, which essentially changes the search window for a stereo match. Figure 3-5 shows the raw disparities for a typical stereo head. The cameras are slightly verged, so a zero disparity plane (where an object appears at the same place in both images) occurs at some finite distance in front of the cameras. If the stereo algorithm is searching 5 disparities, then without any X offset, it will search as shown in the top red arrow, that is, from disparity 0 to disparity 4. By offsetting one image in the X direction by n pixels, the horopter can be changed to go from $-n$ to $5-n$ raw disparities. This search range is indicated by the lower red arrow.

Generally, it is a good idea to set the X offset to compensate for camera vergence or divergence, that is, to set it so that the furthest horopter plane is at infinity. The reason that this is a good idea is because it's usually possible to control how close objects get to the camera, but not how far away. The offset that puts the far horopter plane at infinity is called X_0 . With this offset, a disparity of 0 indicates an infinitely far object.

The horopter can be determined from Equation (1). For example, if the disparity search window is 0-31, the horopter (using the graph above) will be from approximately 1 meter to infinity. The search window can be moved to an offset by shifting the stereo images along the baseline. The same 32 pixel window could be moved to cover 10-41 pixel disparities, with a corresponding horopter of 0.8 meters to 2.2 meters.

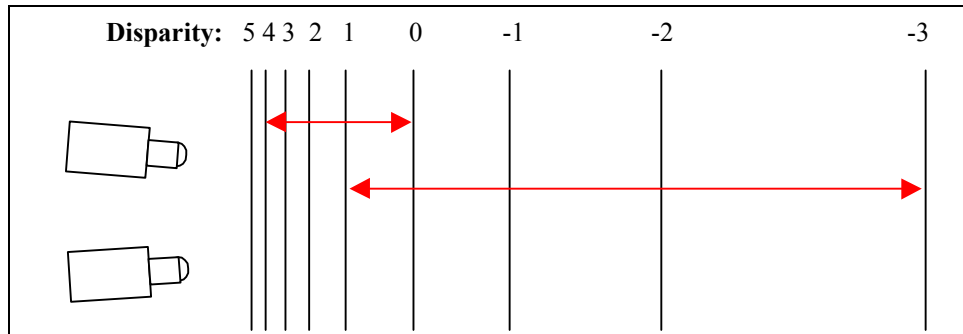


Figure 3-5. Planes of constant disparity for verged stereo cameras. A search range of 5 pixels can cover different horopters, depending on how the search is offset between the cameras.

The location and size of the horopter depends on the application. If an object falls outside the horopter, then its true disparity will not be found, and instead it will get some random distribution of disparities. Figure 3-4 shows what happens when the object's range falls outside the horopter. In the left image, the disparity search window is correctly positioned so that objects from 1 meter to infinity are in view. In the right image, the window has been moved back so that objects have higher disparities. However, close objects are now outside of the horopter, and their disparity image has been "broken up" into a random pattern. This is typical of the disparity images produced by objects outside the horopter.

For a given application, the horopter must be large enough to encompass the ranges of objects in the application. In most cases, this will mean positioning the upper end of the horopter at infinity, and making the search window large enough to see the closest objects.

The horopter is influenced not only by the search window and offset, but also by the camera parameters and the baseline. The horopter can be made larger by some combination of the following:

- Decreasing the baseline.
- Decreasing the focal length (wider angle lenses).
- Increasing pixel width.
- Increasing the disparity search window size.

As the cameras are moved together, their viewpoints come closer, and image differences like disparity are lessened. Decreasing the focal length changes the image geometry so that perceived sizes are smaller, and has a similar effect. It also makes the field of view larger, which can be beneficial in many applications. However, very small focal length lenses often have significant distortion that must be corrected (see the section on calibration). Another way to change the image geometry is to make the pixels wider. This can be done by scaling the image, e.g., from 320x240 to 160x120, which doubles the pixel size. Note that it is only necessary to change the pixel width. Most framegrabbers have hardware scaling to arbitrary resolutions.

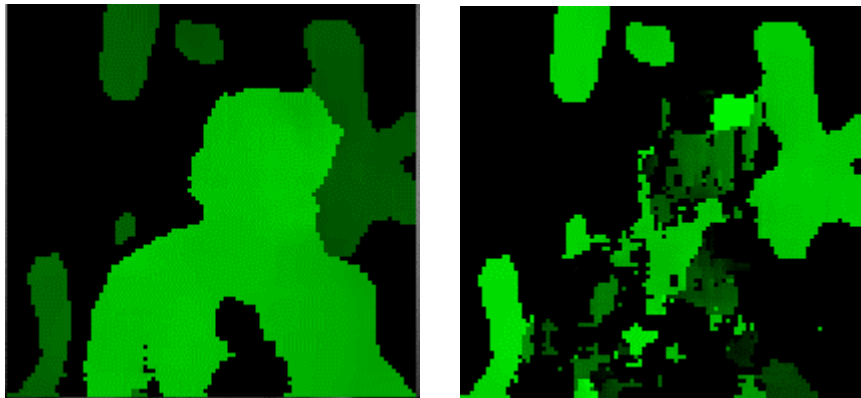


Figure 3-4. Disparity image for all regions within the horopter (left) and some regions outside the horopter (right).

These first three options change the camera geometry, and thus have a corresponding effect on the range resolution, which decreases (see below). The only way to increase the horopter size and maintain range resolution is to increase the disparity search window size, which leads to more computation. Multiresolution methods, which use several sizes of an image, each with its own horopter, are one way to minimize computation (see, for example, the paper by Iocchi and Konolige at www.ai.sri.com/~konolige/svs).

3.3 Range Resolution

Often it's important to know the minimal change in range that stereo can differentiate, that is, the *range resolution* of the method. Given the discussion of stereo geometry above, it's easy to see that that range resolution is a function of the range itself. At closer ranges, the resolution is much better than farther ranges.

Range resolution is governed by the following equation.

$$(2) \Delta r = (r^2/bf) \Delta d$$

The range resolution, Δr , is the smallest change in range that is discernable by the stereo geometry, given a change in disparity of Δd . The range resolution goes up (gets worse) as the square of the range. The baseline and focal length both have an inverse influence on the resolution, so that larger baselines and focal lengths (telephoto) make the range resolution better. Finally, the pixel size has a direct influence, so that smaller pixel sizes give better resolution. Typically, stereo algorithms can report disparities with subpixel precision, which also increases range resolution.

The figure below plots range resolution as a function of range for the STH-MD1 (MEGA-D) stereo head, which has a baseline of 9 cm. The Stereo Engine interpolates disparities to 1/16 pixel, so Δd is $1/16 * 7.5 \text{ um} = 0.08533 \text{ um}$. The range resolution is shown for a sampling of different lens focal lengths. At any object distance, the range resolution is a linear function of the lens focal length.

Equation 2 shows the range resolution of a perfect stereo system. In practice, video noise, matching errors, and the spreading effect of the correlation window all contribute to degrading this resolution.

Range resolution is not the same as range accuracy, which is a measure of how well the range computed by stereo compares with the actual range. Range accuracy is sensitive to errors in camera calibration, including lens distortion and camera alignment errors.

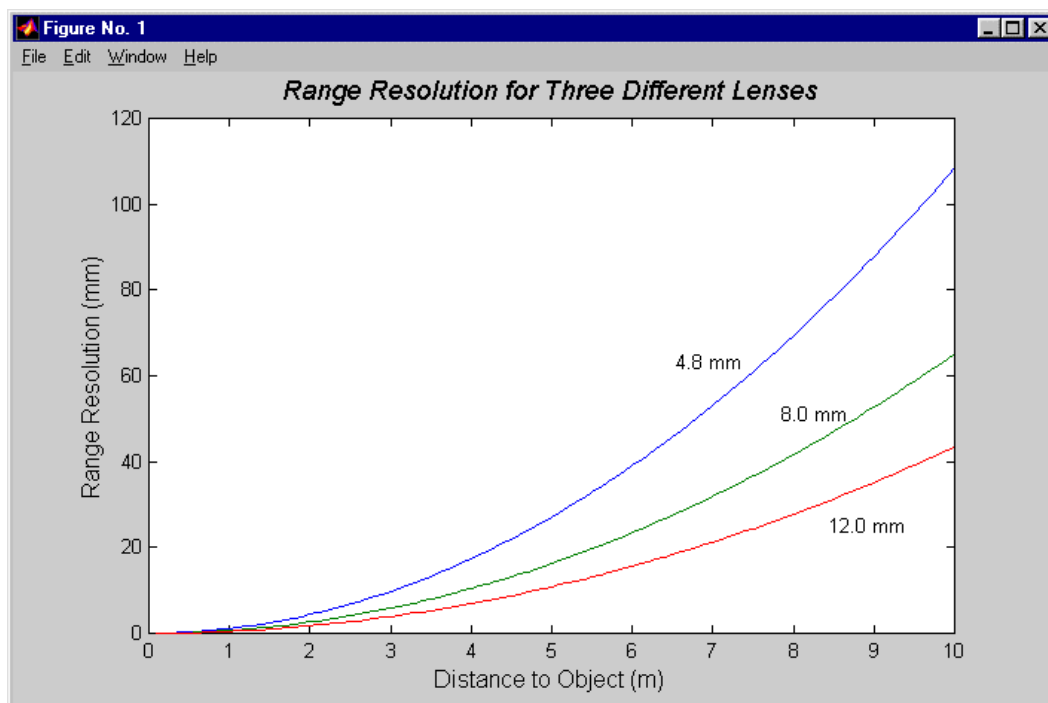


Figure 3-6. Range resolution as a function of range. This plot assumes a baseline of 90 mm, and a pixel size of 7.5 um, with subpixel resolution of 1/16 pixel.

3.4 Area Correlation Window

Stereo analysis is the process of measuring range to an object based on a comparison of the object projection on two or more images. The fundamental problem in stereo analysis is finding corresponding elements between the images. Once the match is made, the range to the object can be computed using the image geometry.

Area correlation compares small patches, or *windows*, among images using correlation. The window size is a compromise, since small windows are more likely to be similar in images with different viewpoints, but larger windows increase the signal-to-noise ratio. Figure 3-7 shows a sequence of disparity images using window sizes from 7x7 to 13x13. The texture filter was turned off to see the effects on less-textured areas, but the left/right check was left turned on.

There are several interesting trends that appear in this side-by-side comparison. First, the effect of better signal-to-noise ratios, especially for less-textured areas, is clearly seen as noise disparities are eliminated in the larger window sizes. But there is a tradeoff in disparity image spatial resolution. Large windows tend to “smear” foreground objects, so that the image of a close object appears larger in the disparity image than in the original input image. The size of the subject’s head grows appreciably at the end of the sequence. Also, in the 7x7 the nose can be seen protruding slightly; at 13x13, it has been smeared out to cover most of the face.

One of the hardest problems with any stereo algorithm is to match very small objects in the image. If an object does not subsume enough pixels to cover an appreciable portion of the area correlation window, it will be invisible to stereo processing. If you want to match small objects, you have to use imagers with good enough spatial resolution to put lots of pixels on the object.

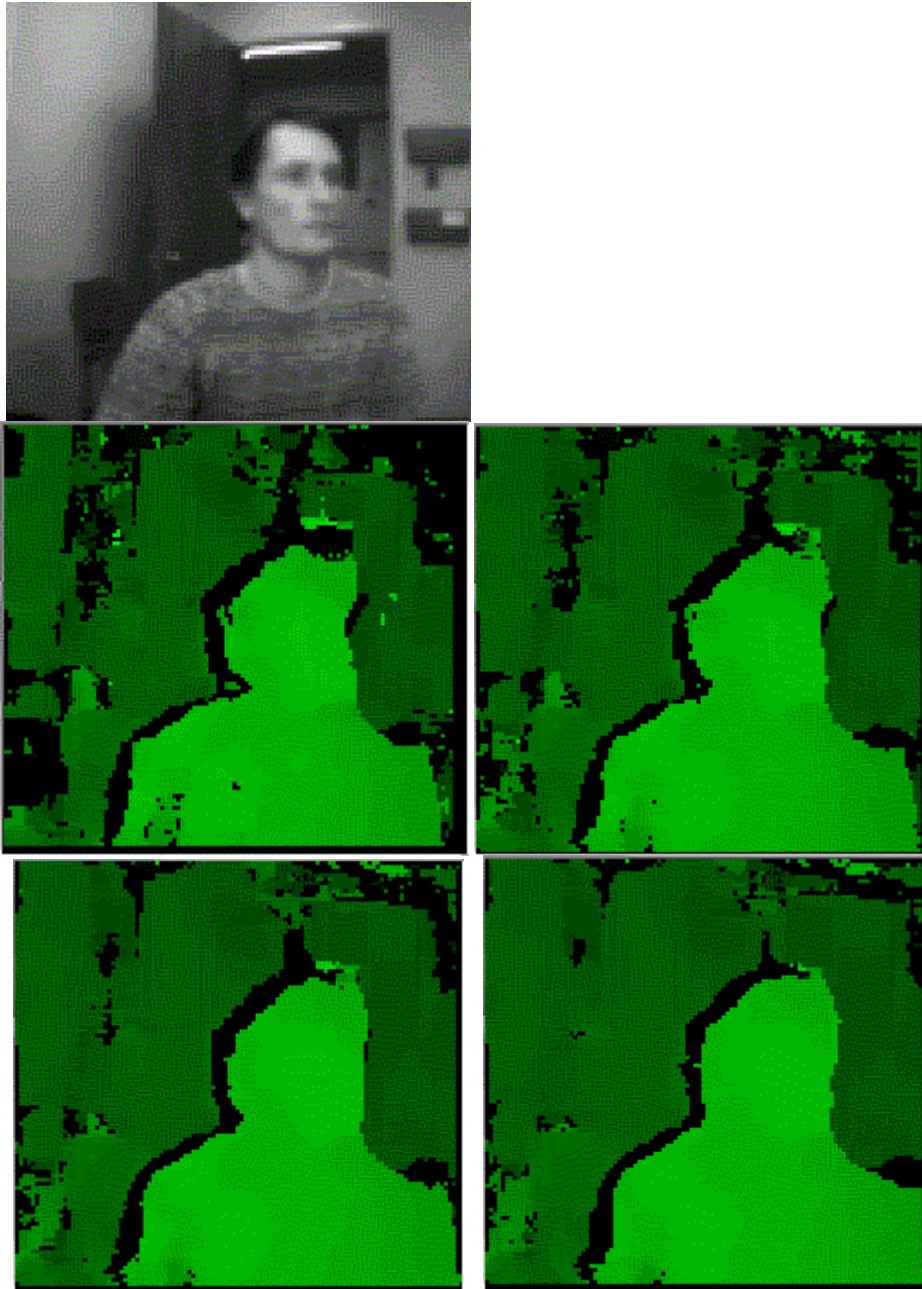


Figure 3-7 Effects of the area correlation window size. At top is the original left intensity image. The greenscale images show windows of 7x7, 9x9, 11x11, and 13x13 windows (clockwise from upper left).

3.5 Multiscale Disparity

Multiscale processing can increase the amount of information available in the disparity image, at a nominal cost in processing time. In multiscale processing, the disparity calculation is carried out at the original resolution, and also on images reduced by 1/2. The extra disparity information is used to fill in dropouts in the original disparity calculation (Figure 3-8).

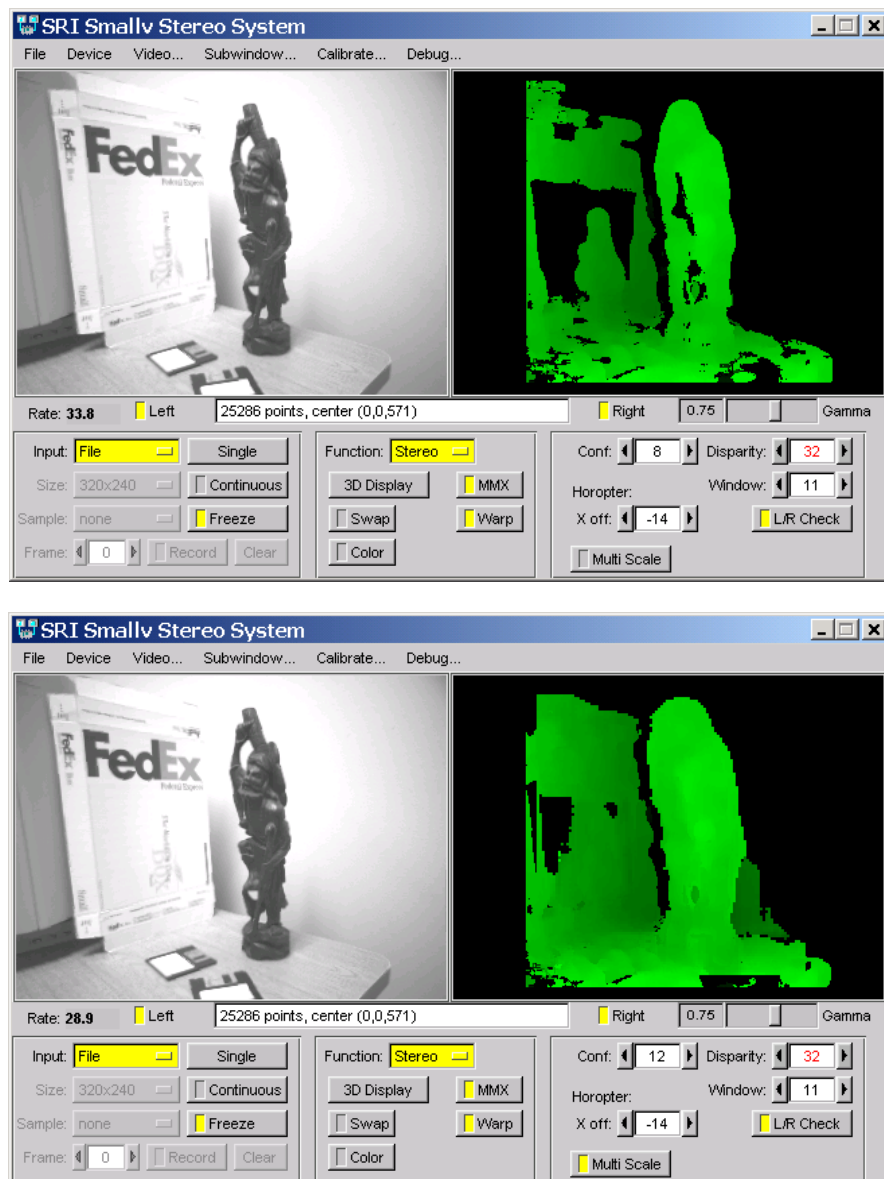
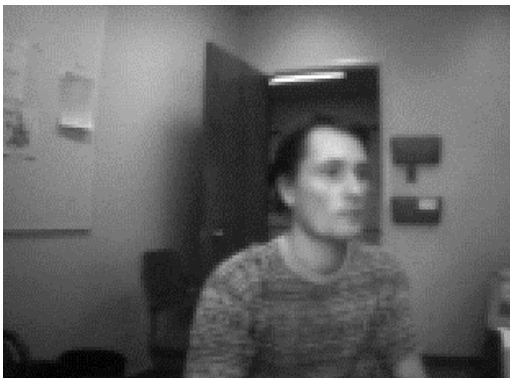


Figure 3-8 Effects of multiscale disparity calculation. Upper figure shows disparity dropouts in a typical scene, where there is not enough texture for correlation to be reliable. Adding disparity information from a 1/2 resolution image (lower part of figure) shows additional coverage in the disparity image.

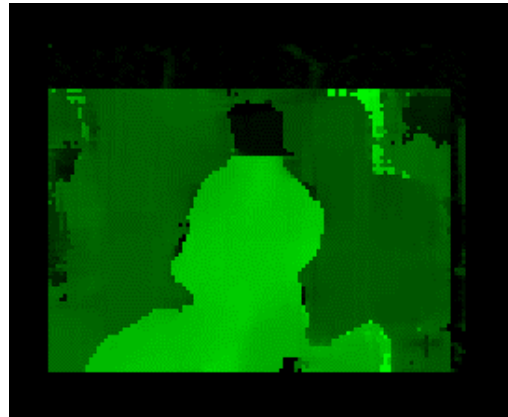
3.6 Filtering

Like most vision algorithms, the results of stereo processing can contain errors. In the case of stereo, these errors result from noisy video signals, and from the difficulty of matching untextured or regularly textured image areas. Figure 3-9 shows a typical disparity image produced by the SRI algorithm. Higher disparities (closer objects) are indicated by brighter green (or white, if this paper is printed without color). There are 64 possible levels of disparity; in the figure, the closest disparities are around 40, while the furthest are about 5. Note the significant errors in the upper left and right portion of the image, where uniform areas make it hard to estimate the disparity.

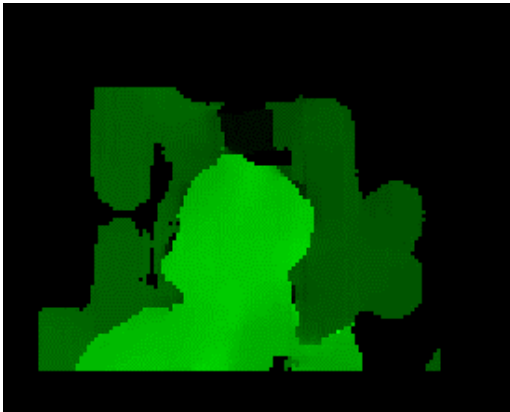
In Figure 3-9(c), the interest operator is applied as a postfilter. Areas with insufficient texture are rejected as low confidence: they appear black in the picture. Although the interest operator requires a threshold, it's straightforward to set it based on noise present in the video input. Showing a blank gray area to the imagers produces an interest level related only to the video noise; the threshold is set slightly above that. Or, more simply, you can use the temporal variance of poorly textured matches to adjust the texture threshold. Observing the disparity image during realtime display, there will usually be areas that flicker rapidly. Adjust the threshold upward until these regions disappear. If there are no such regions, adjust the threshold downward until just before they appear.



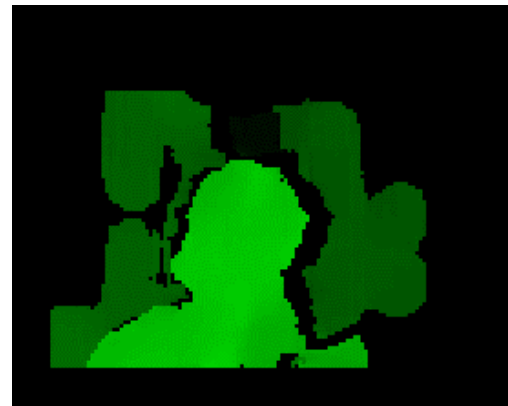
(a) Input grayscale image, one of a stereo pair



(b) Disparity image from area correlation



(c) Texture filter applied



(d) Left/right and texture filter applied

Figure 3-9 Post-filters applied to a disparity image. (c) is a texture filter that eliminates textureless areas. (d) is a consistency check between left and right stereo matches.

There are still errors in portions of the image with disparity discontinuities, such as the side of the subject's head. These errors are caused by overlapping the correlation window on areas with very different disparities. Application of a left/right check can eliminate these errors, as in Figure 3-9(d). The left/right

check can be implemented efficiently by storing enough information when doing the original disparity correlation.

In practice, the combination of an interest operator and left/right check has proven to be the most effective at eliminating bad matches.

3.7 Performance

Using standard PC hardware, running either MS Windows 95/98/ME/2000/NT or Linux, the SVS can compute stereo range in real time. Table 3-1 gives some typical timings for a 500 MHz Pentium III processor. Because the Stereo Engine has a very small memory footprint, the timings scale almost linearly with increasing processor speed. These timings include the complete stereo algorithm detailed above: dewarping of input images, disparity computation and interpolation, and post-filtering using a texture filter and left/right filter.

Frame size	Number of Disparities	Frame Rate
160x120	16	180 Hz
160x120	32	100 Hz
320x240	16	45 Hz
320x240	32	24 Hz
640x480	32	6 Hz

Table 3-1 Processing rates on a Pentium III 500 MHz machine.

4 Calibration

NOTE: There is a Calibration Addendum manual that details the exact steps necessary to perform calibration, and includes troubleshooting information. Please consult that manual for more detailed information about the calibration procedure.

Most stereo camera setups differ from an ideal setup in which the cameras are perfect pinhole imagers and are aligned precisely parallel. The divergence from ideal causes problems in the quality of the stereo match since epipolar lines are not horizontal. In addition, if the camera calibration is unknown, one does not know how to interpret the stereo disparities in terms of range to an object. Camera calibration addresses these issues by creating a mathematical model of the camera.

SVS incorporates a simple automatic procedure for calibration, using a planar object that can be printed on a standard printer. The calibration is preformed by fitting a model to a number of images taken of a planar calibration object. The user presents the object to the stereo rig in five different (arbitrary) poses. The calibration procedure finds model features in the images, and then calculates a best-fit calibration for the rig. The procedure works for many different combinations of imagers, baselines, and lenses, including wide-angle lenses with severe distortion.

When is it necessary to perform calibration? In general, whenever an action changes the camera intrinsics (lens focal length and center axis) or extrinsics (the cameras move with respect to each other). Here are some actions that would necessitate re-calibration:

- Changing lenses
- Screwing the lenses in or out of their mount
- Zooming, if the lenses are zoom lenses
- Changing the baseline of the cameras
- Any movement or rotation of one camera independent of the other, e.g., severe vibration or shock can change the cameras' relative position

A rigid mount that keeps the cameras stable with respect to each other is a necessity for a stereo rig. For example, the MEGA-D uses an extruded aluminum frame to stabilize the cameras. There are some actions that do *not* require re-calibration:

- Changing the lens focus with a focusing ring on the lens
- Changing the lens aperture

The next section reviews the calibration procedure, detailing the steps required to generate a calibration file.

4.1 Calibration Procedure

An automatic calibration procedure using five image pairs of a planar calibration target is included as part of the `smallv` program. Given the image pairs of the calibration object, the system automatically locates corner features in the target, fits a model of the target to the images, and finally produces an estimate for the left and right camera intrinsics, the stereo head extrinsics, and the rectification matrices \mathbf{P}_0 , \mathbf{P}_1 , \mathbf{H}_0 , and \mathbf{H}_1 . These values are then used by `smallv`, and can also be saved as a parameter file for later re-use. More information about the calibration procedure can be found in the Calibration Addendum to the User's Manual.

4.1.1 Calibration procedure steps

1. Create the calibration object. Print out a copy of the file `data/check.pdf` (Figure 4-1) and paste it to a surface that is as flat as possible. We use a wooden cutting board as a backing surface at SRI.
2. Start the `smallv` application and start capturing video. It is recommended that you set the video resolution to at least 320x120 in order to get enough detail of the calibration object. A calibration computed when capturing video at a higher resolution can be used for future video captured at any resolution with the same cameras.
3. Bring up the calibration window by pressing the `Calibrate...` menu button. Fig. 4-4 shows the calibration dialog window (the figure shows the dialog after an image has been captured and processed).
4. Determine the appropriate characteristics of the camera imagers and enter them into the four boxes in the middle of the dialog. If you have one of the Videre Design stereo heads, check the appropriate box and the parameters are loaded automatically.
5. Acquire five stereo pairs of the calibration object at different rotations and translations. Try to avoid views that differ by a simple translation, as they are less informative than views with variation in rotation. As shown in Fig 4-4, there is a tab control that shows only one pair at a time; choose a tab to select another pair. To capture the current video feed into a stereo pair box, simply press the capture button. You can also save and load images to and from disk using the `load` and `save` buttons.

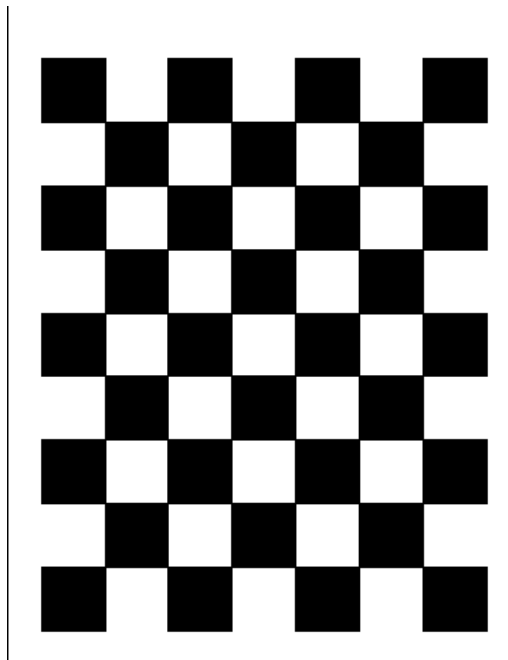


Figure 4-1 Checkerboard calibration object.

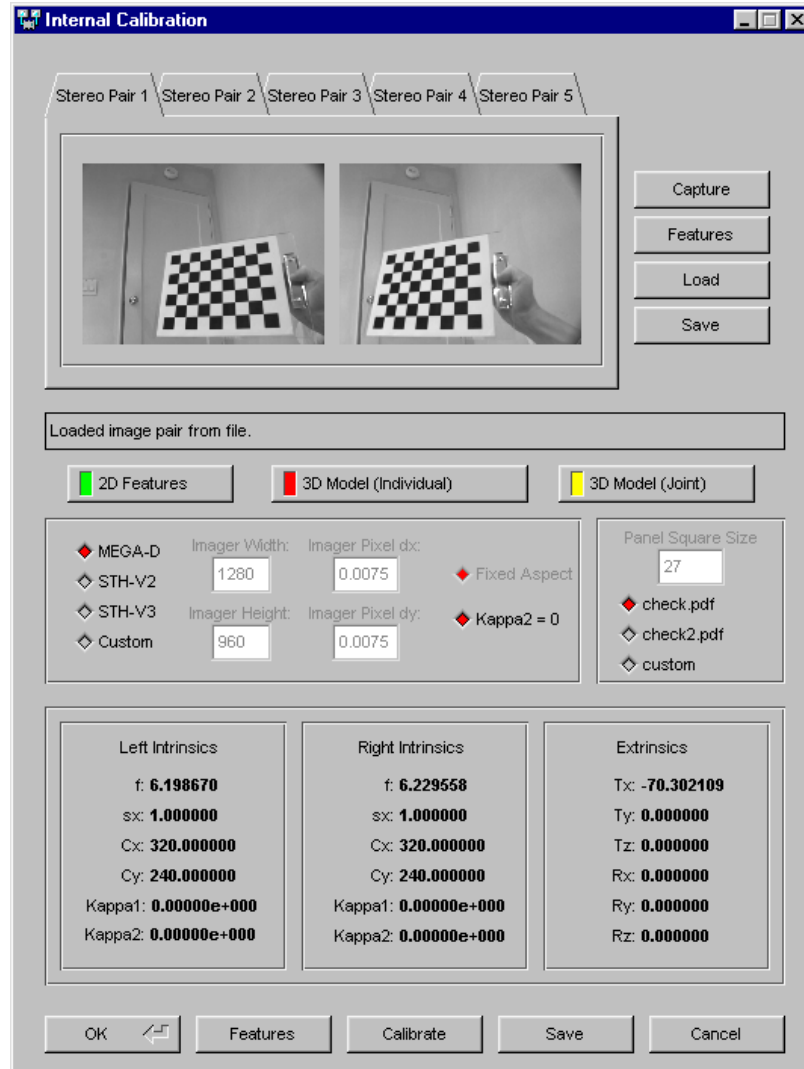


Figure 4-4 Smallv calibration dialog window.

6. Detect the corner features in all views by pressing the `features` button in the lower bar of buttons. This cycles through all the images, displaying the corners in green as they are detected. If the feature finder fails on an image, please re-capture the image and redetect the features. When redetecting features for a single stereo pair, use the `features` button in the stereo pair box.
7. Compute the calibration parameters (intrinsic, extrinsic) and rectification matrices by pressing the `calibrate` button in the lower button bar. This operates in three phases:
 - a) Calibrates individual views using a planar model of the calibration object. The projection of these model features is shown in red.
 - b) Calibrates all the views jointly using nonlinear optimization over all the intrinsic and extrinsic parameters. This phase usually takes a few minutes, and when finished, the projected model features are shown in yellow.
 - c) Computes the rectification matrices from the joint calibration in (b).
8. When the calibration is finished, the parameter listing at the bottom is updated, and you have a couple additional options in the lower button bar: `save` writes the parameter file to disk and `OK` exits the calibration dialog and transfers the new parameters to the main `smallv` window.

4.1.2 Calibration Target

The standard calibration target, `check.pdf`, can be printed out on a single piece of 8.5 x 11 inch paper. In some cases, this image is too small, e.g., when using wide baseline cameras. There is another target, `check2.pdf`, that has squares twice as large as the standard one (54 mm instead of 27 mm). This target must be assembled from four printed pieces of 8.5 x 11 inch paper. It is recommended for large-baseline systems, where the target must be placed at a distance from the cameras in order to be seen in both of them.

4.1.3 Imager Characteristics

The calibration process must be related to the geometry of the camera imagers. There are three important parameters:

1. Pixel size in mm (width and height)
2. The width of the imager in pixels
3. The width of the image output by the framegrabber

The stereo rigs from Videre Design (MEGA-D, STH-V2, and STH-V3) have preset values; just use the appropriate button in the Calibrate dialog. Otherwise, select the `Custom` value, and choose values according to the instructions below.

The pixel size can be found from the specifications of the imager. If you cannot find these, just use defaults of 0.010 mm. The calibration will determine the pixel aspect ratio (width / height). The calibrated lens focal length will not be correct because the pixel scale will be off, but that will not change the validity of the calibration.

The imager width is the number of sels (sensor elements) in each line of the imager. Again, this can be found in the imager specifications. If these are not available, just use the image width as it comes from the framegrabber, e.g., for NTSC video it is 640.

The framegrabber width is the width of the image output by the framegrabber, in pixels. Typically it will be an analog NTSC signal, which is 640 pixels. For digital imagers, such as the MEGA-D, the image size in sels and the framegrabber width are the same.

5 API Reference – C++ Language

With SVS 2.2x, the standard programming interface to the SVS libraries is in C++. To add stereo processing to your own programs, you call functions in the Stereo Engine library. These functions are available in `svs.dll` (Windows 95/98/2000/NT) or `libsv.so` (Unix systems). The header file is `src/svsclass.h`. The current version of the library is 2.2d.

Source code samples for the C++ API are in the directory `samplescpp`. A simple example of the use of these functions is in the sample program `samplescpp/stframe.cpp`.

5.1 Threading and Multiple Stereo Devices

5.1.1 Threading Issues

The SVS core library functions (`svs.dll`, `libsvs.so`) are thread-safe: they can be used in any thread in a process. Of course, the user is responsible for not overlapping calls in different threads, e.g., starting up two competing disparity calculations using the same object in different threads.

The MEGA-D and Dual-DCAM acquisition libraries are also thread-safe, in general. However, there are some known quirks under MS Windows. The most obvious of these is the `Open()` call for the MEGA-D. This call must be made in the main thread. Subsequent accesses using `GetImage()` can be made in any thread.

Graphic window output is handled by the FLTK cross-platform windowing system. This system is not, in general, thread safe. Calls to the FLTK functions must all be made from the same thread; multiple threads are allowed in the application program, as long as all FLTK calls come from the same thread. There is a nascent thread locking mechanism in FLTK, but it is not yet incorporated into SVS.

5.1.2 Multiple Devices

Multiple stereo devices (MEGA-D and Dual-DCAM) can be accessed simultaneously from a single process, or from multiple processes. Only one process may access a given device, using an `Open()` call. Once this call is made, subsequent calls to `Open()` from other processes will fail until the device is released.

Currently, there are several restrictions on multiple device usage. Most of these restrictions apply to the MEGA-D.

1. Under Win32, the MEGA-D reserves most of the bus bandwidth, even if the frame rate is lowered with the `SetRate()` call. Therefore, for the present it is possible to stream video from multiple devices simultaneously only if they are on different IEEE 1394 busses, i.e., attached to separate IEEE 1394 cards. This restriction should be lifted in the near future. There is no such restriction under Linux.
2. When using multiple devices on the same IEEE 1394 bus, it may be necessary to lower the frame rate or frame size before starting streaming video. For example, the Dual-DCAM can use most of the bus bandwidth at 640x480, 15 Hz. Setting the frame rate to 7.5 Hz, or the frame size to 320x240, will allow more devices to be accessed.

5.2 C++ Classes

There are three main classes for SVS: classes that encapsulate stereo images, classes that produce the images from video or file sources, and classes that operate on stereo images to create disparity and 3D images. These classes are displayed in Figure 5-1. The header file is `src/svsclass.h`.

The basic idea is to have one class (`svsStereoImage`) for stereo images and the resultant disparity images, which performs all necessary storage allocation and insulates the user from having to worry about these issues. Stereo image objects are produced from video sources, stored image files, or memory buffers by the `svsAcquireImages` classes, which are also responsible for rectifying the images according to parameters produced by the calibration routines. Disparity images and 3D point clouds are produced by the stereo processing class `svsStereoProcess` acting on stereo image object, with the results stored back in the stereo image object. Finally, display classes allow for easy display of the images within a GUI.

Figure 5-2 shows a simple example of using the classes to produce and display stereo disparity results. The full program example is in `samples/stframe.cpp`. The basic operations are:

1. Make a video source object and open it. Which video source is used depends on which framegrabber interface file has been loaded: see Section 2.1.2.
2. Make a stereo processing object for producing disparity results from a stereo image.
3. Make some display window objects for displaying images and disparity results.
4. Open the video source.
5. Set the frame size and any other video parameters you wish, and read in rectification parameters from a file.
6. Start the video acquisition.
7. Loop:
 - a. Get the next stereo image.
 - b. Calculate disparity results.
 - c. Display the results.

Image source classes	Stereo Image and Parameter classes	Stereo Processing classes	Display classes
<code>svsAcquireImages</code>	<code>svsStereoImage</code>	<code>svsStereoProcess</code>	<code>svsWindow</code>
<code>svsVideoImages</code>			<code>svsGLWindow</code>
<code>svsFileImages</code>	<code>svsImageParams</code>		<code>svsDebugWin</code>
	<code>svsRectParams</code>		
	<code>svsDistParams</code>		

Figure 5-1 SVS C++ Classes

```
// Make a video source object, using the loaded framegrabber interface
svsVideoImages *videoObject = getVideoObject();

// Make a stereo processing object
svsStereoProcess *processObject = new svsStereoProcess();

// Open the video source
bool ret = videoObject->Open();
if (!ret) { ...error code... }

// Read in rectification parameters
videoObject->ReadParams("../data/megad-75.ini");

// Set up display windows
int width = 320, height = 240;
svsWindow *win1 = new svsWindow(width,height);
svsWindow *win2 = new svsWindow(width,height);
win1->show();
win2->show();

// Start up the video stream
videoObject->SetSize(width, height);
ret = videoObject->Start();
if (!ret) { ... error code ... }

// Acquisition loop
while (1)
{
    // Get next image
    svsStereoImage *imageObject = videoObject->GetImage(400);
    if (!imageObject) { ... error code ...}
    // calculate disparity image
    processObject->CalcStereo(imageObject);
    // display left image and disparity image
    win1->DrawImage(imageObject, svsLEFT);
    win2->DrawImage(imageObject, svsDISPARITY);
}
```

Figure 5-2 A simple program for video acquisition and stereo processing. The full program is in `samplescpp/stframe.cpp`.

5.3 Parameter Classes

<code>svsImageParams</code>	Image frame size and subwindow parameters
<code>svsRectParams</code>	Image rectification parameters
<code>svsDispParams</code>	Image stereo processing (disparity) parameters

Parameter classes contain information about the format or processing characteristics of stereo image objects. Each stereo image object contains an instance of each of the above classes. Application programs can read these parameters to check on the state of processing or the size of images, and can set some of the parameters, either directly or through class member functions.

5.3.1 Class `svsImageParams`

Frame size and subwindow parameters for stereo images. In general, the only way these parameters should be changed is through member functions of the appropriate objects, e.g., using `SetSize` in the `svsVideoImages` class.

5.3.2 Class `svsRectParams`

Rectification parameters for stereo images. They are used internally by the rectification functions. Application programs should not change these parameters, and will have few reasons to look at the parameter values. Rectification parameters are generated initially by the calibration procedure, then written to and read from parameter files.

5.3.3 Class `svsDispParams`

Disparity parameters control the operation of stereo processing, by specifying the number of disparities, whether left/right filtering is on, and so on. Most of these parameters can be modified by application programs.

5.4 Stereo Image Class

<code>svsStereoImage</code>	Stereo image class
-----------------------------	--------------------

The stereo image class encapsulates information and data for a single stereo image pair, along with any of its processed results, e.g., disparity image or 3D point cloud.

Stereo image objects are usually produced by one of the image acquisition classes (`svsVideoImages` or `svsFileImages`), then processed further by an `svsStereoProcess` object.

An `svsStereoImage` object holds information about its own state. For example, there are Boolean flags to tell if there is a valid set of stereo images, whether they are rectified or not, if a valid disparity image has been computed, and so on.

The `svsStereoImage` class handles all necessary allocation of buffer space for images. User programs can access the image buffers, but should be careful not to de-allocate them or destroy them.

5.4.1 Constructor and Destructor

```
svsStereoImage () ;  
~svsStereoImage () ;
```

Constructor and destructor for the class. The constructor initializes most image parameters to default values, and sets all image data to NULL.

```
char error[256] ;
```

If a member function fails (e.g., if `ReadFromFile` returns false), then `error` will usually contain an error message that can be printed or displayed.

5.4.2 Stereo Images and Parameters

```
bool haveImages ;           // true if we have good stereo images  
bool haveColor ;           // true if left image color array present  
bool haveColorRight ;      // true if right image color array present  
svsImageParams ip ;        // image format, particular to each object  
unsigned char *Left() ;    // left image array  
unsigned char *Right() ;   // right image array  
unsigned long *Color() ;   // left-color image array  
unsigned long *ColorRight() ; // right-color image array
```

These members describe the stereo images present in the object. If stereo images are present, `haveImages` is true. The stereo images are always monochrome images, 8 bits per pixel. Additionally, there may be a color image, corresponding to the left image, and a color image for the right imager, if requested. Color images are in RGBX format (32 bits per pixel, first byte red, second green, third blue, and fourth undefined). If the left color image is present, `haveColor` is true. The color image isn't used by the stereo algorithms, but can be used in post-processing, for example, in assigning color values to 3D points for display in an OpenGL window. Similarly, if the right color image is present, `haveColorRight` is true. The color images may be input independently of each other.

Frame size parameters for the images are stored in the variable `ip`. The parameters should be considered read-only, with one exception: just before calling the `SetImage` function.

The `Left`, `Right`, and `Color` functions return pointers to the image arrays. User programs should not delete this array, since it is managed by the stereo object.

5.4.3 Rectification Information

```
bool isRectified; // have we done the rectification already?
bool haveRect;   // true if the rectification params exist
svsRectParams rp; // rectification params, if they exist
```

The images contained in a stereo image object (left, right and left-color) can be *rectified*, that is, corrected for intra-image (lens distortions) and inter-image (spatial offset) imperfections. If the images are rectified, then the variable `isRectified` will be true.

Rectification takes place in the `svsAcquireImage` classes, which can produce rectified images using the rectification parameters. The rectification parameters can be carried along with the stereo image object, where they are useful in further processing, for example, in converting disparity images into a 3D point cloud.

If rectification parameters are present, the `haveRect` variable is true. The rectification parameters themselves are in the `rp` variable.

5.4.4 Disparity Image

```
bool haveDisparity; // have we calculated the disparity yet?
svsDisparityParams dp; // disparity image parameters
short *Disparity(); // returns the disparity image
```

The disparity image is computed from the stereo image pair by an `svsStereoProcess` object. It is an array of short integers (signed, 16 bits) in the same frame size as the input stereo images. The image size can be found in the `ip` variable. It is registered with the left stereo image, so that a disparity pixel at X,Y of the disparity image corresponds to the X,Y pixel of the left input image. Values of -1 and -2 indicate no disparity information is present: -1 is for low-texture areas, and -2 is for disparities that fail the left/right check.

If the disparity image has been calculated and is present, then `haveDisparity` is true. The parameters used to compute the disparity image (number of disparities, horopter offset, and so on) are in the parameter variable `dp`.

The disparity image can be retrieved using the `Disparity` function. This function returns a pointer to the disparity array, so it is very efficient. User programs should not delete this array, since it is managed by the stereo object.

5.4.5 3D Point Array

```
bool have3D; // do we have 3D information?
int numPoints; // number of points actually found
float *X(), *Y(), *Z(); // 3D point arrays
```

The 3D point arrays are the 3D points that correspond to each pixel in the left input image. It has the same size (width and height) as the input stereo images. The 3D point array is computed from the disparity image using the external camera calibration parameters stored in `rp`. An `svsStereoProcess` object must be used to compute it.

Each point is represented by a coordinate (X,Y,Z) in a frame centered on the left camera focal point. The Z dimension is distance from the point perpendicular to the camera plane, and is always positive for valid disparity values. The X axis is horizontal and the positive direction is to the right of the center of the

image; the Y axis is vertical and the positive direction is down relative to the center of the image (a right-handed coordinate system). Negative values of Z are used to indicate there was no valid disparity reading at a pixel

If the 3D array is present, then `have3D` is true. The actual number of 3D points present in the arrays is given by `numPoints`.

5.4.6 File I/O

```
bool SaveToFile(char *basename);    // saves images and params to files
bool ReadFromFile(char *basename);  // gets images and params from files
bool ReadParams(char *name);        // reads just params from file
bool SaveParams(char *name);        // save just params to file
```

Images and parameters in a stereo object can be saved to a set of files (`SaveToFile`), and read back in from these files (`ReadFromFile`). The `basename` is used to create a file set. For example, if the `basename` is `TESTIMAGE`, then the files set is:

```
TESTIMAGE-L.bmp    // left image, if present
TESTIMAGE-R.bmp    // right image, if present
TESTIMAGE-C.bmp    // left color image, if present
TESTIMAGE.ini      // parameter file
```

Just the parameters can be read from and written to a parameter file, using `ReadParams` and `SaveParams`. These functions take the explicit name of the file, e.g., `TESTIMAGE.ini`. Parameter files have the extension `.ini`, by convention.

5.4.7 Copying Functions

```
void SetImages(unsigned char *left, // Sets images from user data
               unsigned char *right,
               unsigned char *color,
               unsigned char *color_right,
               svbImageParams *ip = NULL,
               svbRectParams *rp = NULL,
               bool rect = false,
               bool copy = false);
void CopyFrom(svbStereoImage *si); // copies contents of si to object
```

These functions are not used in typical applications, since they manipulate the stereo object buffers. User programs can insert buffer data into a stereo image object using the above functions. These functions are generally useful for making memory buffers of sequences of images, rather than for initial input of images. For example, if you want to input images from your own stereo rig, with images stored in memory, it is recommended to use the `svbStoredImages` acquisition class, which will produce `svbStereoImage` samples. Acquisition classes can perform rectification operations, while the `svbStereoImage` class cannot.

Optional parameter information can be supplied with the images, via the parameter arguments; otherwise, the parameters already present in the object remain the same. If any of the image arguments is `NULL`, then no image data is inserted for that image. If the `copy` argument is `true`, then the buffer contents are copied onto the stereo image object's own buffers. If not, then the input buffers are used internally.

5.5 Acquisition Classes

<code>svsAcquireImages</code>	Base class for all acquisition
<code>svsVideoImages</code>	Acquire from a video source
<code>svsFileImages</code>	Acquire from a file source
<code>SvsStoredImages</code>	Acquire from a memory source

Acquisition classes are used to get stereo image data from video or file sources, and put it into `svsStereoImage` structures for further processing. During acquisition, images can be *rectified*, that is, put into a standard form with distortions removed. Rectification takes place automatically if the calibration parameters have been loaded into the acquisition class.

The two subclasses acquire images from different sources. `svsVideoImages` uses the capture functions in the loaded `svsgrab` DLL to acquire images from a video device such as the MEGA-D stereo head. `svsFileImages` acquires images from BMP files stored on disk.

5.5.1 Constructor and Destructor

```
svsAcquireImages() ;
virtual ~svsAcquireImages() ;
```

These functions are usually not called by themselves, but are implicitly called by the constructors for the subclasses `svsVideoImages` and `svsFileImages`.

5.5.2 Rectification

```
bool HaveRect() ;
bool SetRect(bool on) ;
bool GetRect() ;
bool IsRect() ;
bool ReadParams(char *name) ;
bool SaveParams(char *name) ;
```

These functions control the rectification of acquired images. `HaveRect()` is true when rectification parameters are present; the normal way to input them is to read them from a file, with `ReadParams()`. The argument is a file name, usually with the extension `.ini`. If the acquisition object has rectification parameters, they can be saved to a file using `SaveParams()`.

Rectification of acquired images is performed automatically if `HaveRect()` is true, and rectification processing has been turned on with `SetRect()`. Calling `ReadParams()` will also turn on `SetRect()`. The state of rectification processing can be queried with `GetRect()`.

If the current image held by the acquisition object is rectified, the `IsRect()` function will return true.

5.5.3 Controlling the Image Stream

```
bool CheckParams() ;
bool Start() ;
bool Stop() ;
svsStereoImage *GetImage(int ms) ;
```

An acquisition object acquires stereo images and returns them when requested. These functions control the image streaming process.

`CheckParams()` determines if the current acquisition parameters are consistent, and returns true if so. This function is used in video acquisition, to determine if the video device supports the modes that have been set. If the device is not opened, `CheckParams()` returns false.

`Start()` starts the acquisition streaming process. At this point, images are streamed into the object, and can be retrieved by calling `GetImage()`. `GetImage()` will wait up to ms milliseconds for a new image before it returns; if no image is available in this time, it returns NULL. If an image is available, it returns an `svsStereoImage` object containing the image, rectified if rectification is turned on. The `svsStereoImage` object is controlled by the acquisition object, and the user program should not delete it. The contents of the `svsStereoImage` object are valid until the next call to `GetImage()`.

`Start()` returns false if the acquisition process cannot be started. `Stop()` will stop acquisition.

NOTE: `GetImage()` returns a pointer to an `svsStereoImage` object. This object can be manipulated by the user program until the next call to `GetImage()`, at which point its data can change. The user program should not delete the `svsStereoImage` object, or any of its buffers. All object and buffer allocation is handled by the acquisition system. If the application needs to keep information around across calls to `GetImage()`, then the `svsStereoImage` object or its buffers can be copied.

5.5.4 Error String

char *Error()

Call this function to get a string describing the latest error on the acquisition object. For example, if video streaming could not be started, `Error()` will contain a description of the problem.

5.6 Video Acquisition

The video acquisition classes are subclasses of `svsAcquireImages`. The general class is `svsVideoImages`, which is referenced by user programs. This class adds parameters and functions that are particular to controlling a video device, e.g., frame size, color mode, exposure, and so on.

Particular types of framegrabbers and stereo heads have their own subclasses of `svsVideoImages`. In general, the user programs won't be aware of these subclasses, instead treating it as a general `svsVideoImage` object.

A particular framegrabber interface class is accessed by copying a DLL file to `svsgrab.dll`. For example, for the MEGA-D stereo head and IEEE 1394 interface, copy `svspix.dll` to `svsgrab.dll` (see Section 2.1). Every such interface file defines a subclass of `svsVideoImages` that connects to a particular type of framegrabber and its associated stereo head.

To access the `svsVideoImages` object for the interface file, the special function `svsGetVideoObject()` will return an appropriate object.

5.6.1 Video Object

`svsVideoImages *svsGetVideoObject()`

Returns a video acquisition object suitable for streaming video from a stereo device. The particular video object that is accessed depends on the video interface file that has been loaded; see Section 2.1. This function creates a new video object on each call, so several devices can be accessed simultaneously, if the hardware supports it.

5.6.2 Device Enumeration

**`int Enumerate()`
`char **DeviceIDs()`**

Several MEGA-D and Dual DCAM stereo devices can be multiplexed on the same computer. Any such device connected to an IEEE 1394 card is available to SVS. The available devices are enumerated by the `Enumerate()` function, which returns the number of devices found, and sets up an array of strings that have the identifiers of the devices. The ID array is returned by the `DeviceIDs()` function. The user application should not destroy or write into the array, since it is managed by the video object. The strings are unique to the device, even when plugged into different machines.

The `Enumerate()` function rescans the bus each time it is called, so whenever devices are plugged or unplugged, it can be called to determine which devices are available. `DeviceIDs()` should only be called after at least one `Enumerate()`.

MEGA-D and Dual DCAM devices can be intermixed on the same bus. However, SVS loads only a single video driver, so any particular SVS application will see only the MEGA-Ds or the Dual DCAMs.

5.6.3 Opening and Closing

**`bool Open(char *name = NULL)`
**`bool Open(int devnum)`
`bool Close()`****

Before images can be input from a stereo device, the device must be opened. The `Open()` call opens the device, returning true if the device is available. An optional name can be given to distinguish among several existing devices. The naming conventions for devices depend on the type of device; typically it is a serial number or other identifier. These identifiers are returned by the `Enumerate()` call. Alternatively,

a number can be used, giving the device in the order returned by the `Enumerate()` function, i.e., 1 is the first device, 2 is the second, and so on. A value of 0 indicates any available device.

Upon opening, the device characteristics are set to default values. To set values from a parameter file, use the `ReadParams()` function.

A stereo device is closed and released by the `Close()` call.

5.6.4 Image Framing Parameters

```
bool SetSize(int w, int h)
bool SetSample(int decimation, int binning)
bool SetRate(int rate)
bool SetOffset(int ix, int iy, int verge)
bool SetColor(bool on, bool onr = false)
bool CheckParams()
```

See Sections 2.1.4, 2.1.5, and 2.1.6 for more information on frame sizes and sampling modes.

These functions control the frame size and sampling mode of the acquired image. `SetSize(w,h)` sets the width and height of the image returned by the stereo device. In most cases, this is the full frame of the image. For example, most analog framegrabbers perform hardware scaling, so that almost any size image can be requested, and the hardware scales the video information from the imager to fit that size. In most analog framegrabbers, the sampling parameters (decimation and binning) are not used, and a full-frame image is always returned, at a size given by the `SetSize()` function.

The digital stereo devices allow the user to specify a desired frame rate. Typically the device defaults to the fastest frame rate allowed, and the application can choose a different one to minimize bus traffic. The MEGA-D and Dual-DCAM have different interpretations of the frame rate parameter; see Table 5-1 below.

Some stereo devices, such as the MEGA-D, allow the user to specify a *subwindow* within the image frame. The subwindow is given by a combination of sampling mode and window size. The sampling mode can be specified by `SetSample()`, which sets binning and decimation for the imager. The MEGA-D supports sub-sampling the image at every 1, 2 or 4 pixels; it also supports binning (averaging) of 1 or 2 (a 2x2 square of pixels is averaged). For example, with binning = 2 and decimation = 2, the full frame size is 320 x 240 pixels. Using `SetSize()`, a smaller subwindow can be returned. The offset of the subwindow within the full frame comes from the `SetOffset()` function, which specifies the upper left corner of the subwindow, as well as a *vergence* between the left and right images.

`SetColor()` turns color on the left image on or off. Additionally, some applications require color from the right imager also, and setting the second argument to true will return a color image for the right imager. Generally, returning color requires more bus bandwidth and processing, so use color only if necessary.

The video frame parameters can be set independently, and not all combinations of values are legal.

Device	Rate Parameter	Frame Rate
MEGA-D	0, 1	Normal
	2	Normal / 2
	3	Normal / 3
	4	Normal / 4
Dual-DCAM	30	30 Hz
	15	15 Hz
	7	7.5 Hz

Table 5-1 Frame rates as a function of the `SetRate()` parameter. MEGA-D frame rates are determined from the base rate by clock division. Dual-DCAM rates are determined directly as frames/second.

The `CheckParams()` function returns `true` if the current parameters are consistent.

None of the frame or sampling mode parameters can be changed while images are being acquired, except for the offset parameters. These can be changed at any time, to pan and tilt the subwindow during acquisition.

5.6.5 Image Quality Parameters

```
bool SetExposure(bool auto, int exposure, int gain)  
bool SetBalance(bool auto, int red, int blue)  
bool SetLevel(bool auto, int brightness, int contrast)
```

See Section 2.1.9 and 2.1.12 for more information about video quality parameters.

These functions set various video controls for the quality of the image, including color information, exposure and gain, brightness and contrast. Not all stereo devices support all of the various video modes described by these parameters.

In general, parameters are normalized to be integers in the range [0,100].

`SetExposure()` sets the exposure and gain levels for the device. If `auto` is chosen, the manual parameters are ignored.

`SetBalance()` sets the color balance for the device. Manual parameters for red and blue differential gains are between -40 and 40. If `auto` is chosen, the manual parameters are ignored.

`SetLevel()` sets the brightness and contrast for the device. In `auto` mode, the brightness value is ignored. Contrast is always set manually.

These functions can be called during video streaming, and their effect is immediate.

5.6.6 Controlling the Video Stream

See the functions in Section 5.5.3.

5.7 File and Memory Acquisition

The file and memory acquisition classes are subclasses of `svsAcquireImages`. These classes are used to input stereo images from files, or from arrays in memory, and present them for processing. Users who have their own stereo devices, and acquire images into memory, can use these classes to perform stereo processing with the SVS libraries.

While images are not streamed from files in the same way as from a video source, the function calls are similar. After opening the file, `Start()` and then `GetImage()` function is called to retrieve the stored image. A single image file set is repeatedly opened and read in by successive calls to `GetImage()`.

A file sequence consists of file sets whose basenames end in a 3-digit number. Opening a file set that is part of a sequence causes the rest of the sequence to be loaded on successive calls to `GetImage()`.

5.7.1 File Image Object

```
svsFileImages()  
~svsFileImages()
```

Constructor and destructor for the file acquisition object.

5.7.2 Setting Images from Files

```
bool Open(char *basename)  
bool ReadFromFile(char *basename)  
bool Close()
```

The `Open()` function opens a file set and reads it into the object; see Section 2.2.2 for information about file sets. The file set can include calibration parameters that describe the rectification of the images.

Another file can be read into the file object using the `ReadFromFile()` function. The new images replace the old ones.

The file is closed with the `Close()` function.

5.7.3 Stored Image Object

```
svsStoredImages()  
~svsStoredImages()
```

Constructor and destructor for the file acquisition object.

5.7.4 Setting Images from Memory

```
bool Load(int width, int height,  
           unsigned char *lim, unsigned char *rim,  
           unsigned char *cim = NULL, unsigned char *cimr = NULL,  
           bool rect = false, bool copy = false);
```

The `Load()` function sets images from memory into the acquisition object, after which they can be read out (and optionally rectified) using `GetImage()`. The left and right monochrome images must be loaded; the color images are optional. If the images are already rectified, set `rect` to `true`.

Normally, user images are passed into the acquisition object as pointers; `GetImage()` will output these pointers unless rectification is performed. In this case, the object manages and outputs its own buffers, which the user should not delete.

User images can also be copied into the object's buffers, if `copy` is set to `true`. Here, the object manages all buffers, and the user program should not destroy them.

Typically, a use program will have images stored in memory, and will want to rectify them as part of the stereo process. To do so, first load a parameter file into the acquisition object, using `ReadParamFile`. Next, turn on rectification by calling `DoRect(true)`. Finally, use `Load()` to attach the stored images to the object; calling `GetImage()` will now return the rectified images.

5.8 Stereo Processing Classes

<code>svsStereoProcess</code>	Stereo processing class
<code>svsMultiProcess</code>	Multiscale stereo processing class

The stereo processing classes perform stereo processing on stereo images encapsulated in an `svsStereoImage` object. The results are stored in the stereo image object. All relevant parameters, such as calibration information and stereo parameters, are also part of the stereo image object.

The processing class `svsStereoProcess` handles basic disparity calculation, as well as conversion of the disparity image into 3D points.

The processing class `svsMultiProcess` extends stereo processing to perform multiple scale stereo processing in computing the disparity image. Multiscale processing adds information from stereo processing at reduced image sizes.

5.8.1 Stereo and 3D Processing

```
svsStereoProcess()
~svsStereoProcess()
bool CalcStereo(svsStereoImage *si)
bool Calc3D(svsStereoImage *si)
bool CalcPoint3D(int x, int y, svStereoImage *si,
                 double *X, double *Y, double *Z)
```

`CalcStereo()` calculates a disparity image and stores it in `si`, assuming `si` contains a stereo image pair and its `haveDisparity` flag is false. To recalculate the stereo results (having set new stereo processing parameters), set the `hasDisparity` flag to false and call `CalcStereo()`.

`Calc3D()` calculates a 3D point array from the disparity image of `si`. If `si` does not have a disparity image, then it is first calculated, and then the point array is computed. The point array is stored in the stereo image object, and the `have3D` flag is set.

For some applications, computing the whole 3D array is not necessary; only certain points are needed. In this case, the function `CalcPoint3D()` is provided. This function returns `true` if the disparity at image point `x,y` exists, and puts the corresponding 3D values into the `X,Y,Z` variables. Otherwise, it returns `false` and does not change `X,Y`, or `Z`.

5.8.2 Multiscale Stereo Processing

```
svsMultiProcess()
~svsMultiProcess()
bool doIt
```

This multiscale class computes stereo disparity at the input image resolution, and also at a x2 reduced image size, then combines the results. Multiscale processing adds additional information, filling in parts of the disparity image that may be missed at the higher resolution.

The `svsMultiProcess` class subclasses `svsStereoProcess`, and is used in exactly the same way. The boolean variable `doIt` turns the multiscale processing on or off.

5.9 Window Drawing Classes

<code>svsWindow</code>	Window class for drawing 2D images
<code>svsDebugWin</code>	Window class for printing output

The window drawing classes output 2D stereo imagery to the display. The display window relies on the FLTK cross-platform windowing system (www.fltk.org), and provides basic graphical object drawing in addition to image display. The `svsDebugWin` class is for text output, useful when debugging programs.

It is also possible to output 3D information, especially point clouds formed from the 3D stereo reconstruction functions. This display relies on the OpenGL window capabilities of FLTK. For more information, see the example code in `samples/svsglwin.cpp`.

5.9.1 Class `svsWindow`

This class outputs 2D images to a display window. It can output monochrome, color, and false-color disparity images. In addition, there is an overlay facility for drawing graphical objects superimposed on the image.

`svsWindow` objects will downsize the displayed images to fit within their borders, using factors of 2. For example, a 640x480 image displayed in a 320x200 window will be decimated horizontally by 2 (to 320 columns), and vertically by 4 (to 120 rows). Images smaller than the display window are not upsampled; they are simply displayed in their normal size in the upper-left corner of the window.

Graphical overlays for the window can be drawn using FLTK drawing functions on the window, e.g., lines, circles, etc. `svsWindow` subclasses the FLTK `Fl_Window` class.

```
svsWindow(int x, int y, int w, int h)
~svsWindow()
```

Constructor and destructor. The constructor creates a new `svsWindow` object, displayed at position `x, y` of any enclosing FLTK object, and with width `w` and height `h`. Typically `w` and `h` are multiples of 160x120.

The window will not be visible until `show()` is called on it.

```
DrawImage(svsStereoImage *si, int which = svsLEFT,
void *ovArg = NULL);
ClearImage()
```

Main drawing function. Draws a component of the stereo image object `si`. The argument `which` specifies which component is drawn, according to the following table.

<code>svsLEFT, svsRIGHT</code>	Left and right monochrome images
<code>svsLEFTCOLOR, svsRIGHTCOLOR</code>	Left and right color images
<code>svsDISPARITY</code>	Disparity image (displays in green false color)

The optional last argument is passed to any assigned overlay drawing function (see `svsDrawOverlay` below).

To clear an image from the window, and reset it to black, use `ClearImage()`.

```
virtual DrawOverlay(svsImageParams *ip, void *ovArg);
DrawOverlayFn(void (*fn)(svsWindow *, svsImageParams *, void *ovArg))
```

These functions draw overlay information on the image displayed by `svsWindow`. There are two ways to draw overlays. One is to subclass `svsWindow`, overriding the `DrawOverlay()` function. Then, the subclass can perform any FLTK drawing within the subclass function.

Another way to use overlays is to assign an overlay function to the `svsWindow` object, with `DrawOverlayFn`. This overlay function is called every time the overlay needs to be drawn.

The last argument to these functions is passed in from the argument specified in the `DrawImage()` function.

5.9.2 Class `svsDebugWin`

This class displays text in a scrollable window.

```
svsDebugWin(int x, int y, int w, int h, char *name = NULL)  
~svsDebugWin
```

Constructor and destructor. The `x, y` arguments specify placement of the upper-left corner of the window; `w` and `h` give the width and height. An optional title (`name`) can be given.

The window will not be displayed until the `show()` function is called.

```
Print(char *str)
```

Prints a string on the debug window. Each string is printed on a new line, and the window scrolls to that line.